UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

José Norberto Guiz Fernandes Corrêa

LIFTER - DISPONIBILIZAÇÃO DE APLICAÇÕES VIA *CONTAINERS* DE *SOFTWARE* EM UM *CLUSTER* DE ALTO DESEMPENHO

Florianópolis

2016

José Norberto Guiz Fernandes Corrêa

LIFTER - DISPONIBILIZAÇÃO DE APLICAÇÕES VIA *CONTAINERS* DE *SOFTWARE* EM UM *CLUSTER* DE ALTO DESEMPENHO

Trabalho de Conclusão de Curso submetido ao Programa de Graduação em Sistemas de Informação para a obtenção do Grau de Bacharel em Sistemas de Informação.
Orientador: Prof. Dr. Mario Antonio Ribeiro Dantas

Florianópolis

2016

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

A ficha catalográfica é confeccionada pela Biblioteca Central.

Tamanho: 7cm x 12 cm

Fonte: Times New Roman 9,5

Maiores informaçőes em:
http://www.bu.ufsc.br/design/Catalogacao.html

José Norberto Guiz Fernandes Corrêa

LIFTER - DISPONIBILIZAÇÃO DE APLICAÇÕES VIA *CONTAINERS* DE *SOFTWARE* EM UM *CLUSTER* DE ALTO DESEMPENHO

Trabalho de Conclusão de Curso submetido ao Programa de Graduação em Sistemas de Informação para a obtenção do Grau de Bacharel em Sistemas de Informação.
Orientador: Prof. Dr. Mario Antonio Ribeiro Dantas

Florianópolis

2016

José Norberto Guiz Fernandes Corrêa

LIFTER - DISPONIBILIZAÇÃO DE APLICAÇÕES VIA _CONTAINERS_ DE _SOFTWARE_ EM UM _CLUSTER_ DE ALTO DESEMPENHO

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de Bacharel em Sistemas de Informação, e aprovado em sua forma final pelo Programa de Graduação em Sistemas de Informação.

Florianópolis, 24 de outubro 2016.

Prof. Dr. Renato Cislaghi
Coordenador de Projetos

Banca Examinadora:

Prof. Dr. Márcio Bastos Castro

Prof. Dr. Mario Antonio Ribeiro Dantas
Orientador

Prof. Dr. Roberto Willrich

À minha família, por todo amor, motivação e exemplo que me deram desde sempre.

AGRADECIMENTOS

A todos da SeTIC/UFSC, pelas oportunidades criadas e ideias fomentadas.

Ao Professor Mario Dantas, pelos norteamentos e auxílios essenciais ao desenvolvimento deste trabalho.

À minha família e amigos, pela presença e ajuda em todos os momentos.

"Each day we go to our work in the hope of discovering, — in the hope that some one, no matter who, may find a solution of one of the pending great problems, — and each succeeding day we return to our task with renewed ardor; and even if we are unsuccessful, our work has not been in vain, for in these strivings, in these efforts, we have found hours of untold pleasure, and we have directed our energies to the benefit of mankind."

Nikola Tesla

RESUMO

Em diversas áreas científicas e profissionais, é crescente a demanda de recursos computacionais de alto desempenho, uma vez que este é essencial na solução de problemas cada vez mais complexos. Porém, o compartilhamento destes recursos por diversos grupos de usuários faz com que diferentes aplicações compartilhem um mesmo ambiente computacional, geralmente criando situações de conflito entre si. Com a modernização de tecnologias de *containers* de software, em especial do *Docker*, tornou-se possível uma abordagem mais flexível na disponibilização de aplicações. Estas tecnologias permitem a criação de ambientes isolados, eliminado os possíveis conflitos entre aplicações, além de possuírem outras funcionalidades que facilitam a gerência de aplicações em uma diversa gama de infraestruturas computacionais. Através do *Docker*, este trabalho utilizou-se desta tecnologia em um *cluster* de alto desempenho da SeTIC/UFSC, de forma a reduzir os recursos e o tempo despendido na disponibilização de aplicações em relação à solução de fábrica utilizada anteriormente neste ambiente. Com a utilização do *Docker*, e tendo como base a solução de fábrica, foram obtidas reduções próximas de 94% no espaço em disco utilizado para armazenamento das aplicações, e de 78% a 96% no tempo despendido para disponibilização destas aplicações no *cluster*.

Palavras-chave: *Containers Linux*, *Cluster*, *Docker*, *HPC*

ABSTRACT

In many scientific and professional fields, the demand for high performance computing resources is ever-increasing, since it is essential in the solution of even more complex problems. However, sharing these resources with many groups of users results in distinct applications sharing the same computing environment, often creating conflict situations among them. With the recent development in software container technologies, the main one being Docker, a more flexible approach to distribute software became possible. These technologies allow the creation of isolated environments, eliminating possible conflicts between applications, and also provide useful features to ease the application management in a wide variety of computing infrastructures. The present work employed this technology, through the use of Docker, in a high-performance cluster at SeTIC/UFSC. The goal was the reduction of the time spent deploying applications to this environment, when compared with the previous factory solution. With the deployment of Docker, and using the factory solution data as a baseline, it was possible to obtain near 94% of disk space usage reduction for the application storage, and a range of 78% to 96% reduction in the time spent deploying applications to the cluster.

Keywords: Linux containers, Cluster, Docker, HPC

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Em diversas áreas científicas e profissionais (tais como engenharia, biologia, meteorologia e astronomia, entre outras), é crescente a demanda de recursos computacionais de alto desempenho, uma vez que estes são essenciais na solução de problemas cada vez mais complexos. Porém, devido ao seu alto custo de aquisição e manutenção, tais ambientes computacionais são geralmente compartilhados entre diversos grupos de pesquisa, de modo a reduzir o custo total entre si. Assim, diferentes aplicações, com requisitos e finalidades distintas, passam a coexistir dentro destes ambientes. No entanto, mesmo para um número pequeno de grupos de usuários, surgem problemas oriundos deste compartilhamento de recursos: conflitos de dependências entre aplicações, indisponibilidade de recursos, problemas de usabilidade, entre outros. Com a modernização de tecnologias de *containers* de software, tornou-se possível uma abordagem mais flexível na disponibilização de aplicações.

De forma a tratar os problemas encontrados em um *cluster* de alto desempenho da SeTIC/UFSC, principalmente o conflito de dependências e o elevado tempo despendido na disponibilização de aplicações, este trabalho consistiu na implementação de uma tecnologia de *containers* de *software* neste ambiente.

Os principais autores que embasaram este trabalho foram Jacobsen e Canon (2015), Dantas (2005) e Meffe, Mussi e Mello (2006).

1.1 OBJETIVOS

Este trabalho teve como objetivo geral a configuração e implementação de uma ferramenta de gerenciamento de *containers* de *software* em um ambiente de *cluster* de alto desempenho. A contribuição pretendida deste trabalho é a de reduzir o tempo de disponibilização de aplicações no *cluster* e facilitar a administração deste ambiente, de forma a obter um melhor aproveitamento dos recursos computacionais existentes.

Os objetivos específicos deste trabalho foram:

1. Realização de um estudo de arquiteturas distribuídas;

2. Realização de um estudo de tecnologias de *containers* de *software*;
3. Configuração do *cluster* de alto desempenho para suporte a *containers* de *software*;
4. Disponibilização de aplicações-exemplo;
5. Avaliação dos ambientes prévio e posterior.

1.2 MÉTODO DE PESQUISA

Este trabalho caracterizou-se como uma pesquisa aplicada, pois objetivou gerar conhecimentos para aplicação prática dirigidos à solução de problemas específicos de um *cluster HPC* da SeTIC/UFSC. As etapas deste trabalho foram divididas em:

- Etapa 1: Análise da fundamentação teórica;
- Etapa 2: Revisão do estado da arte;
- Etapa 3: Configuração do ambiente de *cluster*;
- Etapa 4: Disponibilização de aplicações e coleta de dados;
- Etapa 5: Discussão dos resultados obtidos.

1.3 ESTRUTURA DO TRABALHO

A estrutura deste trabalho é composta dos seguintes capítulos:

- Capítulo 2 - Fundamentação Teórica: é apresentada uma visão geral dos principais conceitos de computação distribuída e de *containers* de *software*, assuntos onde este trabalho está inserido.
- Capítulo 3 - Trabalhos Relacionados: são apresentadas as produções mais recentes e relevantes dentro do escopo deste trabalho.
- Capítulo 4 - Projeto *Lifter*: é relatado o trabalho desenvolvido, apresentando-se o ambiente computacional, a proposta, a implementação e os resultados obtidos.
- Capítulo 5 - Conclusão: são apresentadas as considerações finais e as dificuldades encontradas, além de sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 COMPUTAÇÃO DISTRIBUÍDA

Segundo Dantas (2005), define-se a computação distribuída como um segmento da Ciência da Computação que tem como objetivo a melhoria do desempenho de aplicações distribuídas e paralelas, utilizando-se de complexas infraestruturas computacionais. É um conceito que engloba, de acordo com Tittel (2010), múltiplos sistemas computacionais trabalhando sobre um determinado problema, sendo este problema dividido em várias partes, cada parte solucionada por computadores distintos, que se comunicam entre si através de uma rede.

Para Coulouris, Dollimore e Kindberg (2005), os sistemas distribuídos, sob a visão da arquitetura de máquinas, são considerados configurações com grande capacidade de escalabilidade, pela possibilidade de agregação dos computadores existentes nas redes convencionais em um sistema único, onde a homogeneidade ou heterogeneidade do conjunto de máquinas permite a formação de diferentes configurações.

2.1.1 *Clusters* e Processamento de Alto Desempenho

Tittel (2010) define *cluster* de computadores como um agrupamento interconectado de nós computacionais, operando colaborativamente na execução de aplicações especiais e coordenados por um *software* gerenciador. Além disso, outras razões são consideradas por organizações no uso destes tipos de estruturas computacionais, tais como:

- Redução no tempo para a solução, principalmente em simulações, modelos ou previsões de larga escala e complexas;
- Aproveitamento máximo do investimento já feito em recursos computacionais;
- Estabelecimento de controle das tarefas (também chamadas de *jobs*) e priorização de trabalhos mais importantes para um processamento mais ágil;
- Melhoria geral da confiança, manutenção e disponibilidade computacional por um custo reduzido.

O processamento de alto desempenho (*HPC - High-Performance Computing*) é descrito por Eadline (2009) como uma classe de supercomputadores e *clusters* que solucionam problemas avançados e de alta intensidade computacional. Com o objetivo de aumentar a vazão computacional, um *cluster HPC* é, geralmente, utilizado de duas maneiras:

- Executando concorrentemente diversas tarefas, conforme a Figura 1. É recomendado quando se necessita rodar *jobs* idênticos com parâmetros ou conjuntos de dados distintos. Dependendo dos recursos computacionais existentes, os *jobs* podem precisar aguardar em uma fila o término daqueles que estão em execução. Neste modo, o processamento de um *job* é feito localmente em um nó do *cluster*, ou seja, não existe comunicação com outros nós; além disso, como os *jobs* são independentes, também não existe interação entre eles.

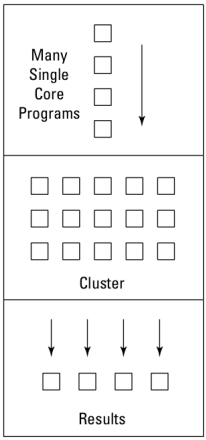

Figura 1 – Processamento concorrente de *jobs* em um *cluster* (EADLINE, 2009)

- Dividindo um grande *job* em subtarefas e executar cada subtarefa em nós distintos do *cluster*, conforme a Figura 2. Este procedimento é realizado a nível de *software* e, portanto, as aplicações necessitam ser modificadas para se utilizarem desta funcionalidade. O método mais comum para desenvolvimento de programas paralelos é a utilização de troca de mensagens com a biblioteca MPI (*Message Passing Interface*). Como as subtarefas comunicam-se

entre si, o tráfego de rede pode ser intenso, de modo que redes de alto desempenho são necessárias para lidar com o alto fluxo de dados.

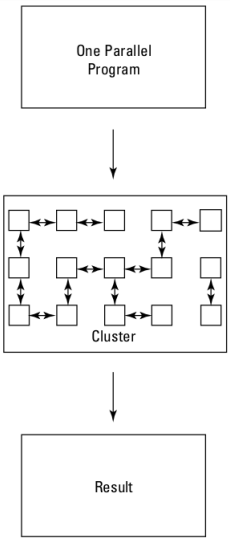

Figura 2 – Processamento paralelo de um *job* em um *cluster* (EADLINE, 2009)

O gerenciamento técnico de *clusters*, segundo Meffe, Mussi e Mello (2006), envolve as tarefas básicas de instalação, configuração, manutenção, monitoramento e escalonamento de recursos. Um ambiente compartilhado por múltiplos projetos ou grupos de usuários necessita de priorização e alocação de recursos eficientes, de forma a garantir resultados para todos os envolvidos. De acordo com Eadline (2009), quanto mais um negócio ou organização depende dos seus ambientes de *clusters* para finalidades de processamento, produção ou receita, mais essencial torna-se a gerência técnica.

Para Tittel (2010), a usabilidade de um *cluster* para os usuários consiste basicamente das aplicações que este suporta, de modo que a facilidade em disponibilizar estas aplicações agiliza o atendimento às demandas dos mesmos. As aplicações utilizadas atualmente em *HPC* abrangem vários setores, tais como:

- Engenharias: design 2D e 3D, modelagem, simulações, verifica-

ções;

- Economia e finanças: análises de risco, predições;
- Geociências: exploração de gás e petróleo;
- Meteorologia: modelagem climática, previsão;
- Biociências: detecção e prevenção de doenças, desenvolvimento de medicamentos;
- Criação de conteúdo digital: processamento digital, renderização;
- Universidades e governos: pesquisa básica e aplicada.

A lista torna-se cada vez maior, pois mais áreas descobrem no processamento de alto desempenho uma ferramenta extremamente útil para a melhor compreensão das suas ciências, mercados e produtos.

2.2 *CONTAINERS* DE SOFTWARE

De acordo com Yu (2007), a virtualização a nível de sistema operacional é definida como a existência de múltiplas instâncias isoladas de espaços de usuário (ao invés de apenas uma), que são gerenciadas pelo *kernel* deste sistema. Estas instâncias, chamadas de *containers* de software (ou simplesmente *containers*), representam todo um ambiente de execução: uma aplicação, juntamente com todas suas dependências, bibliotecas e arquivos de configuração, agrupados em um único "pacote". Ao criar um *container* de uma determinada aplicação e suas dependências, a diversidade de sistemas operacionais e infraestruturas subjacentes são abstraídas.

Máquinas virtuais e *containers* possuem funcionalidades similares quanto ao isolamento e alocação de recursos computacionais. No entanto, uma abordagem arquitetural distinta, ilustrada na Figura 3, permite que os *containers* possuam uma maior portabilidade e eficiência.

Na ilustração anterior, três aplicações em *containers* rodam em um único sistema operacional e compartilham o *kernel* do sistema entre si. Isso faz com que os *containers* sejam mais leves e consumam menos recursos do que máquinas virtuais. A parte compartilhada é somente leitura, enquanto que cada *container* possui o seu espaço para escrita. No entanto, conforme Rubens (2015), *containers* e máquinas virtuais

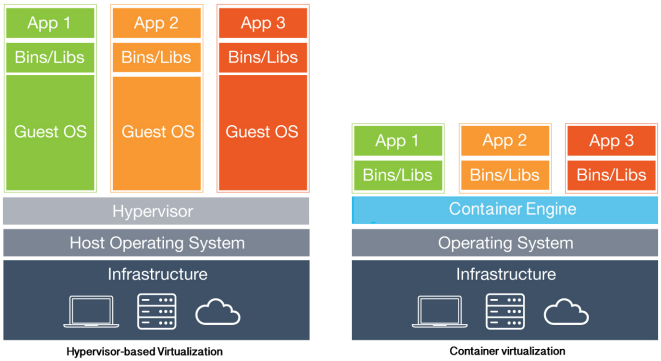

Figura 3 – Diagrama estrutural de máquinas virtuais e *containers* (TWISTLOCK, 2016)

podem ser vistas entre si como tecnologias complementares ao invés de competitivas. Isso se deve ao fato de ser possível rodar *containers* em máquinas virtuais, facilitando, através da virtualização de *hardware*, a gerência da infraestrutura (tais como rede e armazenamento), além de aumentar o isolamento e a segurança.

As raízes da tecnologia de *containers* de *software* datam do final da década de 1970. De acordo com Gunaratne (2016), os principais marcos no histórico desta tecnologia são:

- ***chroot*:** A ideia por trás dos *containers* iniciou em 1979 com o chroot (*change root*) do UNIX. É uma chamada do sistema operacional UNIX que altera o diretório raiz de um processo e seus processos-filho para um novo local do sistema de arquivos, visível apenas para este processo. Esta funcionalidade fornece um espaço de disco isolado para cada processo.

- ***FreeBSD Jails*:** Desenvolvido em 2000, é uma das primeiras tecnologias de *containers* de *software* existentes. É uma chamada de sistema similar ao chroot, porém com funcionalidades adicionais de proteção de processos, tais como isolamento de sistema de arquivos, usuários e rede. Como resultado, é possível associar um endereço IP para cada *jail* e personalizar instalações de *software* e configurações.

- ***Linux VServer*:** Desenvolvido em 2001, é um mecanismo que

pode ser utilizado para particionamento de recursos (sistema de arquivos, tempo de *CPU*, endereços de rede e memória).

- ***Solaris Containers*:** Desenvolvido em 2004, é uma combinação de controladores de recursos de sistema e separação por zonas. As zonas comportam-se como servidores virtuais completamente isolados, dentro de uma única instância do sistema operacional.
- ***OpenVZ*:** Desenvolvido em 2005, faz uso de um *kernel Linux* modificado com a finalidade de fornecer virtualização, isolamento, gerenciamento de recursos e pontos de verificação.
- ***Process Container/cgroups*:** Desenvolvido em 2006 pela Google, possibilita a limitação e isolamento de recursos de um conjunto de processos. Foi renomeado posteriormente para *Control Groups* (cgroups).
- ***LXC*:** Desenvolvido em 2008, é a primeira e mais completa implementação de um gerenciador de *containers Linux*. Tendo como base o uso do cgroups e *namespaces* do *Linux*, é disponibilizado na biblioteca liblxc e possui APIs para diversas linguagens de programação. Em comparação com outras tecnologias de *containers*, o LXC funciona nativamente com o *kernel Linux*.
- ***LMCTFY* (*Let Me Contain That For You*):** Desenvolvido em 2013, é a versão de código-aberto da implementação de *containers Linux* da *Google*, visando garantia de desempenho e *containers* com baixo *overhead*. Em 2015, a *Google* migrou conceitos e abstrações do *LMCTFY* para o projeto *libcontainer*, e por isso não é mais mantido.
- ***Docker*:** Desenvolvido em 2013, é um projeto de código aberto que automatiza a disponibilização de aplicações dentro de *containers* de software. Ele inclui um modelo de imagens em camadas eficiente, repositórios de imagens, interface *CLI*, entre outras funcionalidades. Informações mais detalhadas são apresentadas na seção 2.2.1.
- ***rkt (Rocket)*:** Desenvolvido em 2014, é uma iniciativa desenvolvida pelo *CoreOS*, motivada a fornecer uma segurança mais rigorosa para *containers* em ambientes de produção.
- ***Windows Containers*:** Desenvolvido em 2016, é uma iniciativa da *Microsoft* para adicionar o suporte a *containers* de aplicações *Windows* ao sistema operacional *Windows Server 2016*.

2.2.1 *Docker*

O *Docker* é uma plataforma de disponibilização de aplicações que utiliza funcionalidades de isolamento de recursos do *kernel Linux* para a criação de *containers* independentes. O suporte a *namespaces* do *kernel* isola a visão de uma aplicação em relação ao sistema operacional, incluindo árvore de processos, rede, UIDs e sistemas de arquivos, enquanto que o *cgroups* fornece limitação de recursos, tais como *CPU*, memória, *I/O* e rede. A sua estrutura básica é ilustrada na Figura 4.

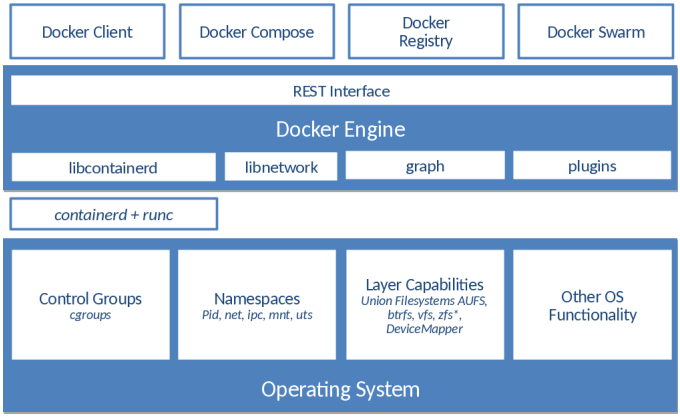

Figura 4 – Visão estrutural do *Docker* (BROWN; STARK, 2016)

Os componentes principais da plataforma *Docker*, de acordo com Jernigan (2016) e Turnbull (2016), são:

- ***Docker Engine*:** O *daemon* que gerencia imagens, *containers*, volumes, redes, entre outros. É construído sobre dois elementos principais:
  - ***runC*:** é uma ferramenta *CLI* que interage com as funcionalidades do *kernel Linux* com o objetivo de instanciar e rodar *containers*.
  - ***containerd*:** é um *daemon* que utiliza o *runC* para gerenciar *containers* e expor suas funcionalidades através de uma *API*.

- ***Docker Client*:** É o componente que interage com o *daemon* do *Docker Engine*.
- ***Dockerfile*:** É um arquivo de texto simples, composto por uma sequência de comandos, que funciona como uma "receita" para a criação de *Docker Images*.
- ***Docker Images*:** A base da criação de *containers*, uma imagem é composta da união de camadas de sistemas de arquivos, empilhadas umas sobre as outras, conforme a Figura 5.
- ***Docker Registry*:** São repositórios de imagens.
- ***Docker Containers*:** Os *containers* propriamente ditos, instâncias criadas a partir de imagens.
- ***Docker Compose*:** Possibilita a criação conjunta de grupos de *containers* para um propósito comum.
- ***Docker Swarm*:** Possibilita a orquestração de *clusters* de *containers*, permitindo a execução escalável de aplicações.

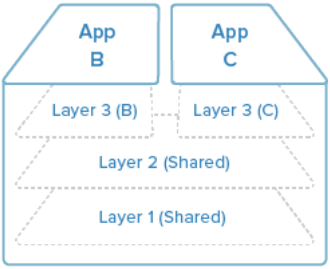

Figura 5 – Diagrama de camadas de uma imagem *Docker* (ELLINGWOOD, 2015)

Tanto o *Docker* quanto as demais soluções existentes possuem limitações quanto à interoperabilidade entre diferentes arquiteturas de processadores e sistemas operacionais: imagens criadas para uma arquitetura *x86-64* não funcionarão nativamente para arquiteturas *ARM* ou *ARM64*, por exemplo, assim como imagens criadas para um sistema operacional *Linux* não funcionarão nativamente para sistemas operacionais *Windows*. É possível o uso de emuladores (como o *QEMU*) ou tradutores (como o *Wine*), de modo a lidar com estas limitações, porém o seu desempenho não é garantido para todos os casos.

3 TRABALHOS RELACIONADOS

Como suporte para a realização deste trabalho, foi realizada uma pesquisa de produções relevantes nas bases *IEEE Xplore Digital Library*, *ACM Digital Library* e *Google Scholar* envolvendo os termos "*container*", "*cluster*" e "*HPC*". Desta busca, foram selecionadas obras apresentadas na Tabela 1. Na seção a seguir, são comentados alguns dos resultados relevantes destas obras para o desenvolvimento deste trabalho.

Tabela 1 – Trabalhos selecionados

Título	**Autoria**
Contain this, unleashing Docker for HPC	Jacobsen e Canon (2015)
Is container-based technology a winner for high performance scientific applications?	Adufu, Choi e Kim (2015)
How advanced cloud technologies can impact and change HPC environments for simulation	Mancini e Aloisio (2015)
An updated performance comparison of virtual machines and Linux containers	Felter et al. (2015)
Orchestrating Docker Containers in the HPC Environment	Higgins, Holmes e Venters (2015)
HPC Made Easy: Using Docker to Distribute and Test Trilinos	Deal (2016)
An introduction to Docker for reproducible research	Boettiger (2015)
Comparison of Virtualization and Containerization Techniques for High-Performance Computing	Zhou et al. (2015)

O trabalho de Jacobsen e Canon (2015) relatou a implementação da infraestrutura de *HPC* baseada em *containers* no *NERSC* (*National Energy Research Scientific Computing Center*), um centro geren-

ciado pelo *Lawrence Berkeley National Laboratory* e pelo *Department of Energy Office of Science*. O protótipo desenvolvido pelo *NERSC*, chamado *Shifter*, foi uma maneira escalável de disponibilizar *containers* em um ambiente de *HPC*. O protótipo converte imagens *Docker* e máquinas virtuais em um formato comum, e este formato permite a distribuição das imagens nos supercomputadores *Cray* da organização. A interface do *Shifter* permite ao usuário selecionar uma imagem de um repositório privado ou do *Docker Hub* e, posteriormente, submeter tarefas ao ambiente, que são executadas inteiramente dentro de *containers*. O fluxo de trabalho deste protótipo é ilustrado na Figura 6.

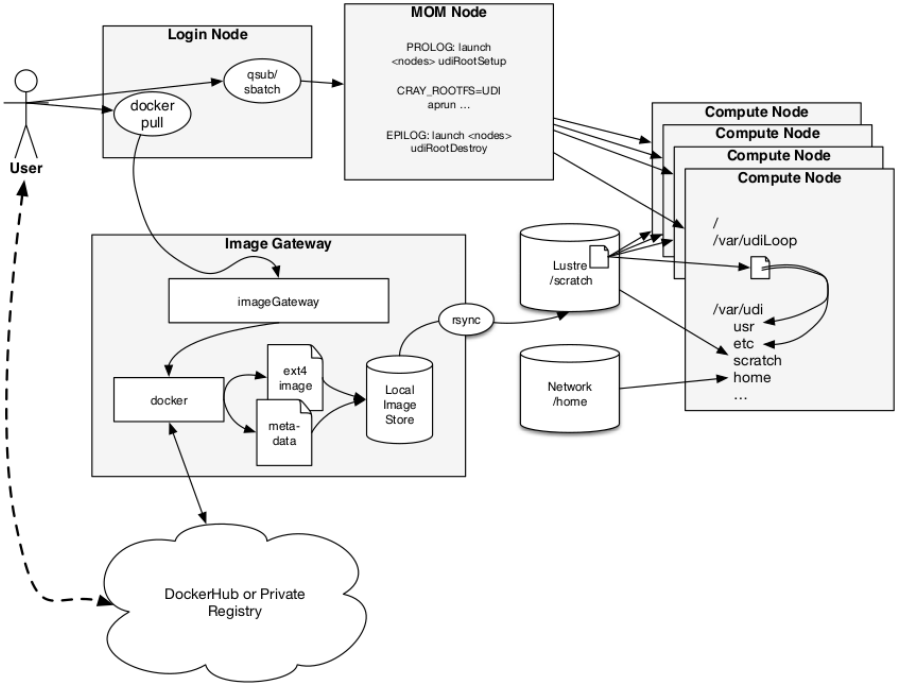

Figura 6 – Fluxo de trabalho do *Shifter* (JACOBSEN; CANON, 2015)

O objetivo do trabalho foi solucionar a rápida expansão das aplicações, bibliotecas e ferramentas demandadas pelas comunidades de usuários deste ambiente. Em diversos casos, os requisitos de duas comunidades distintas conflitavam (distribuições de *Linux* distintas, ver-

sões diferentes de compiladores, etc). As principais necessidades dos usuários consistiam em poder executar suas aplicações científicas no mesmo contexto em que eram desenvolvidas e, em alguns casos, poder facilmente migrar do seu ambiente *desktop* para o ambiente de *HPC*. Algumas comunidades tentaram buscar na computação em nuvem esta flexibilidade, porém encontraram desafios, tais como gerenciar sistemas de arquivos e provisionar recursos, que são normalmente resolvidos por um *cluster* gerenciado. A utilização de *containers* neste contexto permitiu a flexibilidade dos sistemas em nuvem, juntamente com um desempenho próximo do *hardware* nativo.

A implementação de um ambiente de *HPC* com suporte a *containers* de *software* trouxe flexibilidade e facilidade de uso para os seus usuários, com praticamente nenhuma degradação de *performance* em relação ao desempenho nativo na maioria dos casos. Ainda que a tecnologia de *containers* seja relativamente recente nas comunidades científicas e de *HPC*, as vantagens apresentadas podem fazer com que sua adoção seja acelerada nos próximos anos.

No trabalho de Adufu, Choi e Kim (2015), a alocação eficiente de recursos, o tempo de execução de aplicações e a demora em disponibilizar novas aplicações em ambientes de *HPC* foram as dificuldades identificadas. As autoras investigaram o uso de *containers Docker* na execução de uma aplicação científica de alto desempenho (*Autodock3*, um software de modelagem molecular), comparando o desempenho com máquinas virtuais (*VMs*).

Os resultados demonstraram que os tempos de execução utilizando *containers* são menores quando comparados com máquinas virtuais, conforme a Figura 7, principalmente nos tempos de inicialização. Os experimentos também demonstram que o *Docker* gerencia os recursos de memória mais eficientemente quando quantidades maiores que a memória física são alocadas para as instâncias em execução, conforme a Figura 8.

O trabalho de Deal (2016) utilizou o *Docker* para avaliar o desempenho paralelo de testes utilizando o projeto *Trilinos*, um *framework* de *software* orientado a objetos que visa o desenvolvimento de algoritmos e tecnologias para solucionar problemas científicos e de engenharia. Os resultados da execução do *Epetra BasicPerfTest* em um *cluster* e em *containers* mostraram que a diferença de *performance* é praticamente inexistente. Ao executar em paralelo com *MPI*, os resultados foram mais irregulares, apesar de os *containers* terem um desempenho consistente, próximo ao *bare-metal*. Para tarefas com um

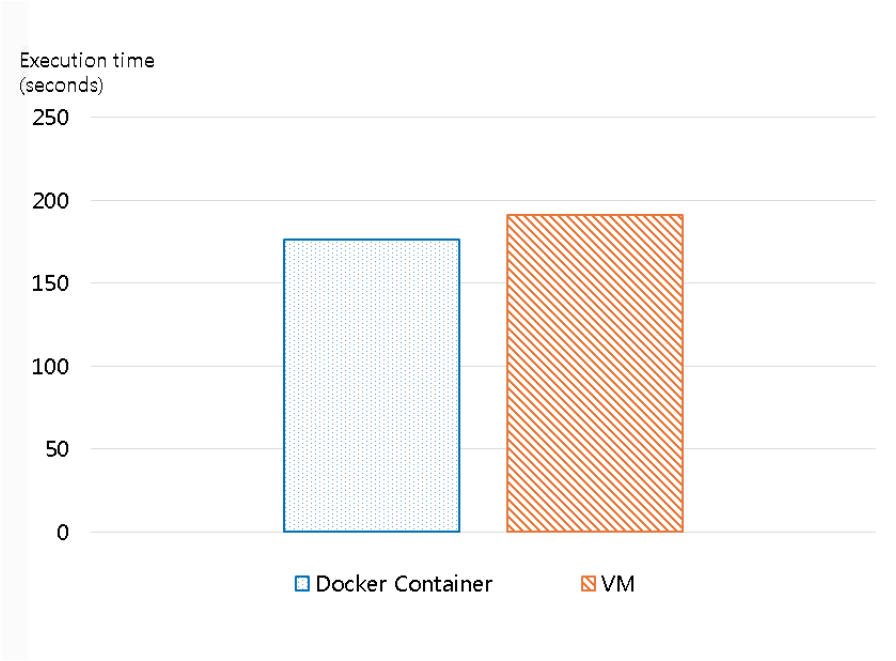

Figura 7 – Comparativo entre *containers* e *VMs* (ADUFU; CHOI; KIM, 2015)

maior número de processos (a partir de 48), os *containers* superaram o desempenho nativo, o que sugere que o uso de *containers* consegue otimizar a execução de aplicações escaláveis.

Dentre os trabalhos selecionados, o produzido por Jacobsen e Canon (2015) foi o que mais se aproximou das dificuldades encontradas no *cluster HPC* da SeTIC/UFSC (apresentado no capítulo a seguir) e dos resultados práticos almejados para este ambiente. Resultados similares também foram apresentados no trabalho de Higgins, Holmes e Venters (2015).

Já os trabalhos de Adufu, Choi e Kim (2015), Deal (2016), Zhou et al. (2015) e Felter et al. (2015) compararam o desempenho de aplicações em diferentes estruturas (*containers*, *bare-metal* e máquinas virtuais), onde os resultados dos mesmos justificam, de maneira quantitativa, o uso de *containers* para aplicações científicas.

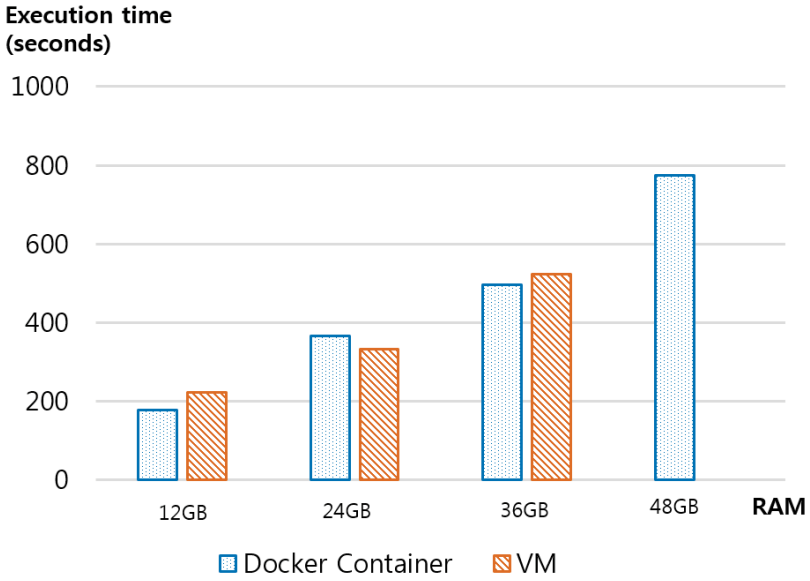

Figura 8 – Comparativo entre diferentes configurações de memória (ADUFU; CHOI; KIM, 2015)

O trabalho de Boettiger (2015) destaca a vantagem no uso do *Docker* em vários ambientes computacionais, principalmente como uma forma de facilitar a reprodução de trabalhos científicos. Como estes ambientes computacionais possuem naturalmente uma evolução rápida, a repetição e extensão de experimentos diversos torna-se um grande desafio.

Por fim, o trabalho de Mancini e Aloisio (2015) descreve como soluções de virtualização e *cloud computing*, adotadas por diversas empresas e organizações nos últimos anos, vêm sendo também adotadas pela área de *HPC*.

3.1 CONSIDERAÇÕES

Todos as obras mencionadas anteriormente foram fundamentais para o desenvolvimento deste trabalho, pois tornaram plausíveis a im-

plementação e a utilização de *containers* em um ambiente de *cluster HPC*. De um modo geral, estas obras embasaram e justificaram a realização do Projeto *Lifter*, apresentado no próximo capítulo.

4 PROJETO *LIFTER*

4.1 AMBIENTE DE TRABALHO

Este trabalho foi realizado em um *cluster* de alto desempenho da SeTIC/UFSC, composto por um servidor *SGI Altix XE250* (*head node*), um servidor *SGI Altix XE210*, quatro servidores *SGI Altix XE310* (dois nós de processamento por *blade*) e um servidor *SGI Altix XE320* (dois nós de processamento por *blade*), interconectados por interfaces de rede *Gigabit Ethernet* através de dois *switches* dedicados, conforme ilustrado na Figura 9. As especificações do *cluster* são apresentadas na Tabela 2.

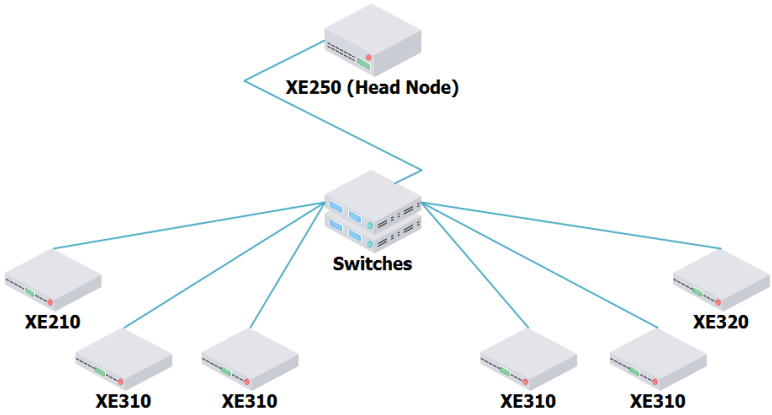

Figura 9 – *Cluster HPC* SeTIC/UFSC

O *cluster* é atualmente utilizado por um número muito reduzido de grupos de pesquisa da UFSC, justamente pela dificuldade em agregar demandas distintas neste mesmo ambiente. Para remediar a situação, a resolução de conflitos entre aplicações deste ambiente era tratada com uma solução da própria fabricante do *cluster* (*SGI*), consistindo de diversos *scripts* (*cinstallman*, *cpower* e outros), que automatizavam a criação de imagens completas (*dumps*) de discos e disponibilizavam estas imagens para os nós de processamento via *boot* PXE (*Preboot eXecution Environment*). Apesar de solucionar o problema inicial, outros

Tabela 2 – Especificações do *cluster HPC* SeTIC/UFSC

Servidor	**Nome**	***CPU***	***RAM***	**Disco**	**Sistema Operacional**	***Kernel***
XE250	admin	2 x Intel Xeon E5420	24GB	1TB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n001	2 x Intel Xeon E5420	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n002	2 x Intel Xeon E5420	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n003	2 x Intel Xeon E5420	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n004	2 x Intel Xeon E5420	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n005	2 x Intel Xeon E5345	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n006	2 x Intel Xeon E5345	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE320	n007	2 x Intel Xeon E5462	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE320	n008	2 x Intel Xeon E5462	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n009	2 x Intel Xeon E5345	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n010	2 x Intel Xeon E5345	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE210	n011	2 x Intel Xeon E5140	16GB	1.5TB	*SUSE Linux Enterprise 12*	4.1.27-27

tornaram-se aparentes:

- Necessidade de reiniciar nós do *cluster* para utilização de aplicações distintas;
- Demora no *boot* via *PXE* nos nós do *cluster*, tempo em que os recursos ficam indisponíveis;
- Consumo de espaço em disco elevado para armazenamento das imagens.

4.2 PROPOSTA

O projeto *Lifter* foi desenvolvido tendo em vista os problemas identificados anteriormente e tendo como base os trabalhos relacionados encontrados. Ele propõe o uso de *containers* de modo a reduzir o tempo para disponibilização, reduzir o uso de espaço em disco e garantir o isolamento das aplicações no *cluster HPC* de estudo, possibilitando o seu uso por um número maior de usuários da comunidade acadêmica da UFSC.

No âmbito deste trabalho, a ferramenta gerenciadora de *containers* utilizada foi o *Docker*, por ser um *software* livre (licença *Apache 2.0*), possuir maior estabilidade e documentação de qualidade, liderar os esforços na criação de um padrão aberto de *containers* e ter uma comunidade ativa de usuários e desenvolvedores. O *Rocket* também foi considerado, e apesar dos avanços recentes em seu desenvolvimento, ainda é um projeto que carece de maturidade para uso em um ambiente de produção.

O *Lifter* consistiu na instalação e configuração do *Docker* em todos os nós do *cluster*, na criação de um repositório de imagens no *head node* e na disponibilização destas imagens para os demais nós. Nesta estrutura, apresentada na Figura 10, o próprio repositório de imagens é disponibilizado dentro de um *container* rodando no *head node*. A transferência de uma determinada imagem é realizada apenas uma vez para cada nó, e esta imagem é armazenada localmente no mesmo. Uma vez transferidas as imagens, é possível instanciar em cada nó os *containers* destas aplicações.

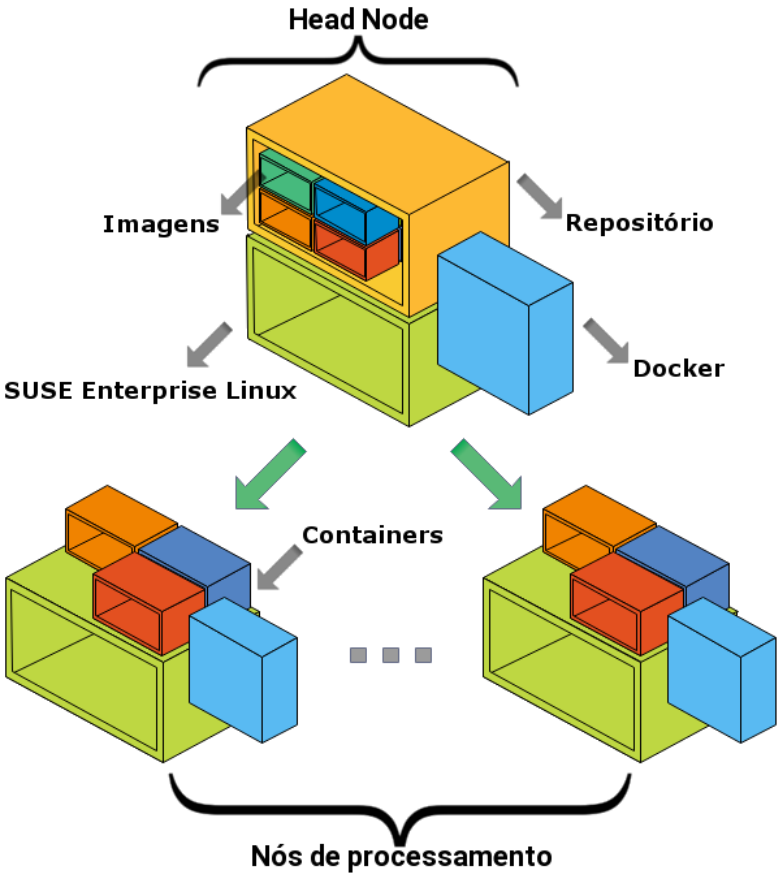

Figura 10 – Estrutura do *Lifter*

Para efeito de comparação pré e pós-projeto, as aplicações selecionadas para testes, já utilizadas no ambiente de trabalho, foram as seguintes:

- ***EnergyPlus*:** É um programa de código aberto (licença *BSD-3-like*), desenvolvido pelo Departamento de Energia dos Estados Unidos, voltado para simulações térmicas e elétricas, utilizado por engenheiros, arquitetos e pesquisadores para auxiliar na modelagem de processos de aquecimento, resfriamento, ventilação e iluminação de edifícios.
- ***Gromacs (GROningen MAchine for Chemical Simulations)*:** É um pacote de código aberto (licença *LGPL*), desenvolvido pela Universidade de Groningen (Holanda), dedicado a cálculos de dinâmica molecular, tais como simulações de proteínas, lipídios, ácidos nucleicos e polímeros.
- ***OpenFOAM*:** É um programa de código aberto (licença *GPL*), desenvolvido pela *OpenCFD Ltd.* e *OpenFOAM Foundation*, dotado de ferramentas para análise numérica e voltada para solução de problemas complexos relacionados a dinâmica de fluidos envolvendo reações químicas, turbulência, transferência de calor, acústica, mecânica dos sólidos e eletromagnetismo.

4.3 IMPLEMENTAÇÃO

Visando a obtenção de resultados mais fidedignos, foi utilizada a versão mais recente do *Docker* (1.12.1) na época em que o trabalho foi realizado. Foi necessária a instalação manual desta versão em todos os nós do *cluster*, pois a versão presente nos repositórios oficiais era obsoleta. Também foram atualizados os *kernels* dos nós para uma versão estável e recente (*Linux* 4.1.27). Informações adicionais quanto as configurações do Docker podem ser visualizadas no excerto de linha de comando do Anexo A.

Para a criação das imagens das aplicações, foram utilizados *Dockerfiles* disponíveis no *Docker Hub*, apresentados nos Anexos B, C e D. As imagens Docker das aplicações foram compiladas e armazenadas em um *Docker Registry* no *head node* do *cluster*. Este repositório roda também dentro de um *container* e é acessível diretamente pelos nós de processamento através da porta 5000 (TCP). A decisão em utilizar um repositório privado surgiu pela necessidade de ter um maior controle

sobre o local onde as imagens são armazenadas: apesar das aplicações utilizadas como exemplo neste trabalho serem de licença aberta, algumas aplicações proprietárias que possam vir a ser utilizadas no *cluster* (tais como *MATLAB* ou *ANSYS*) não podem ser disponibilizadas em um repositório público como o *Docker Hub*.

4.4 CENÁRIOS DE TESTE E RESULTADOS

Uma primeira avaliação é a comparação do espaço ocupado em disco das imagens (*dumps*) previamente utilizadas no *cluster* com as imagens *Docker*, apresentadas resumidamente na Tabela 3.

Tabela 3 – Espaço em disco ocupado por imagens

Aplicação	**Imagens/*Dumps* de Disco**	**Imagens *Docker***	**Diferença**
EnergyPlus 8.5.0	14637MB	268MB	-98.17%
GROMACS 5.0.4	14935MB	976MB	-93.37%
OpenFOAM 2.3.0	16296MB	1525MB	-90.64%
Total:	45868MB	2769MB	-93.96%

Conforme a Tabela 3, o tamanho das imagens *Docker* representam uma redução de espaço em disco próxima de 94% em relação à solução da fabricante, que utiliza *dumps*.

A segunda avaliação consistiu no tempo necessário para a disponibilização das imagens. Foram consideradas duas situações, de modo a se obter um limite mínimo e máximo:

- Tempo de disponibilização para apenas um nó do *cluster*;
- Tempo de disponibilização para todos os nós do *cluster*, paralelamente.

Para cada uma das situações, foram realizadas três medidas e calculada a média. Os resultados obtidos são apresentados nas Tabelas 4 e 5.

Conforme as Tabelas 4 e 5, obteve-se uma redução na faixa de 78% a 96% no tempo de disponibilização utilizando *containers*. Vale ressaltar também o fato de que, além do tempo reduzido, os nós permaneceram ativos e disponíveis para execução de jobs; o que não ocorre com a solução de fábrica do *cluster*, que faz com que os nós fiquem indisponíveis de 21 a 38 minutos enquanto os servidores são reiniciados.

Tabela 4 – Tempo para disponibilização de imagens em um nó

Aplicação	**Imagens/*Dumps* de Disco**	**Imagens *Docker***	**Diferença**
EnergyPlus 8.5.0	1280s (21min 20s)	41s	-96.80%
GROMACS 5.0.4	1351s (22min 31s)	145s (2min 25s)	-89.27%
OpenFOAM 2.3.0	1442s (24min 2s)	315s (5min 15s)	-78.16%

Tabela 5 – Tempo para disponibilização de imagens no *cluster*

Aplicação	**Imagens/*Dumps* de Disco**	**Imagens *Docker***	**Diferença**
EnergyPlus 8.5.0	1998s (33min 18s)	77s (1min 17s)	-96.15%
GROMACS 5.0.4	2051s (34min 11s)	218s (3min 38s)	-89.37%
OpenFOAM 2.3.0	2275s (37min 55s)	473s (7min 53s)	-79.21%

4.5 CONSIDERAÇÕES

De acordo com os resultados obtidos, é notável que a solução de *containers*, adotada no *Lifter*, é uma alternativa muito mais ágil e leve do que a solução de fábrica do *cluster HPC* da SeTIC/UFSC. Esta abordagem, em contraste com a solução antiga, possibilita:

- A inclusão de mais aplicações no *cluster* e, consequentemente, a possibilidade de expandir a quantidade de usuários que utilizam o ambiente;
- O aumento da disponibilidade dos recursos computacionais para execução de *jobs*;
- O encapsulamento das aplicações e a mobilidade entre ambientes computacionais distintos ao delegar aos usuário a criação das imagens das aplicações.

Os resultados significativos, obtidos com o uso de imagens *Docker*, é devido ao fato destas imagens armazenarem apenas do conteúdo relacionado à própria aplicação, enquanto que as imagens da solução de fábrica armazenam, além da aplicação, todo o sistema operacional. Isso acaba influenciando tanto no tamanho das imagens quanto no tempo necessário para transferência das imagens aos nós do *cluster*.

5 CONCLUSÃO

Com o aumento da necessidade por recursos computacionais de alto desempenho, as soluções já consolidadas podem não ser mais suficientes para se atender a demanda. A velocidade das pesquisas e negócios é cada vez maior, e não há espaço para ineficiências nos recursos utilizados. A adoção de tecnologias modernas de *containers* de software em ambientes de *HPC*, conforme este trabalho buscou mostrar, pode contribuir na solução de algumas destas dificuldades.

O trabalho conseguiu atingir o seu objetivo em reduzir o tempo dispendido na disponibilização de aplicações em um *cluster* de alto desempenho da SeTIC/UFSC, além de reduzir a utilização de espaço em disco das imagens de aplicações em relação à solução utilizada anteriormente. Não foram realizados testes intensivos de modo a verificar a diferença de desempenho entre os ambientes pré e pós-*containers*, pois outros trabalhos relacionados já investigaram esta questão em ambientes similares. As ferramentas utilizadas neste trabalho são baseadas na experiência do autor em áreas paralelas (como desenvolvimento de software e gerência de infraestrutura de TI), e representam apenas uma fração das opções disponíveis atualmente.

Algumas das principais dificuldades encontradas na realização deste trabalho foram devidas ao próprio hardware legado do *cluster*, datado de 2008. Os problemas concentraram-se na falha de alguns componentes (como fontes de alimentação e placas de rede), além da utilização de redes *Gigabit Ethernet*, consideradas de baixo desempenho para aplicações *HPC* modernas. No entanto, com o desenvolvimento deste e de trabalhos futuros, espera-se garantir uma sobrevida para este ambiente, que ainda possui um poder de processamento considerável e suficiente para atender várias demandas da comunidade acadêmica da UFSC.

Trabalhos futuros poderão ser realizados com o uso de ferramentas alternativas (como o Rocket), otimizações na configuração do *kernel Linux* e gerenciador de *containers*, e o desenvolvimento de uma ferramenta de gerenciamento de *containers* e imagens de aplicações no *cluster* via interface Web, por exemplo, visando facilitar a interação dos usuários com o ambiente e a execução de aplicações e, consequentemente, reduzir o tempo ocioso da estrutura existente.

REFERÊNCIAS

ADUFU, T.; CHOI, J.; KIM, Y. Is container-based technology a winner for high performance scientific applications? In: IEEE. *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*. Busan, Coréia do Sul, 2015. p. 507–510.

BOETTIGER, C. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, ACM, v. 49, n. 1, p. 71–79, 2015.

BROWN, T.; STARK, J. *Windows Server and Docker - The Internals Behind Bringing Docker and Containers to Windows*. 2016. Disponível em: <http://www.slideshare.net/Docker/windows-server-and-docker-the-internals-behind-bringing-docker-and-containers-to-windows-by-taylor-brown-and-john-starks/>.

COULOURIS, G. F.; DOLLIMORE, J.; KINDBERG, T. *Distributed systems: concepts and design*. Londres, Reino Unido: Pearson Education, 2005.

DANTAS, M. A. R. *Computação distribuída de alto desempenho: redes, clusters e grids computacionais*. Rio de Janeiro, Brasil: Axcel Books, 2005.

DEAL, S. J. *HPC Made Easy: Using Docker to Distribute and Test Trilinos*. 2016.

EADLINE, D. *High Performance Computing for Dummies*. Indianápolis, Estados Unidos: Wiley, 2009.

ELLINGWOOD, J. *The Docker Ecosystem: An Introduction to Common Components*. 2015. Disponível em: <https://www.digitalocean.com/community/tutorials/the-docker-ecosystem-an-introduction-to-common-components/>.

FELTER, W. et al. An updated performance comparison of virtual machines and linux containers. In: IEEE. *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*. Filadélfia, Estados Unidos, 2015. p. 171–172.

GUNARATNE, I. *Evolution of Linux Containers and Future*. 2016. Disponível em: <http://imesh.io/evolution-of-linux-containers-and-future/>.

HIGGINS, J.; HOLMES, V.; VENTERS, C. Orchestrating docker containers in the hpc environment. In: SPRINGER. *International Conference on High Performance Computing*. Bengaluru, Índia, 2015. p. 506–513.

JACOBSEN, D. M.; CANON, R. S. Contain this, unleashing docker for hpc. *Proceedings of the Cray User Group*, 2015.

JERNIGAN, T. *Docker 1.11 et plus: Engine is now built on runC and containerd*. 2016. Disponível em: <https://medium.com/@tiffanyfayj/docker-1-11-et-plus-engine-is-now-built-on-runc-and-containerd-a6d06d7e80ef>.

MANCINI, M.; ALOISIO, G. How advanced cloud technologies can impact and change hpc environments for simulation. In: IEEE. *High Performance Computing & Simulation (HPCS), 2015 International Conference on*. Amsterdã, Holanda, 2015. p. 667–668.

MEFFE, C.; MUSSI, E. O. d. P.; MELLO, L. R. d. *Guia de Estruturação e Administração do Ambiente de Cluster e Grid*. 1ª. ed. Brasília, Brasil: Secretaria de Logística e Tecnologia da Informação, 2006.

RUBENS, P. *What are containers and why do you need them?* 2015. Disponível em: <http://www.cio.com/article/2924995/enterprise-software/what-are-containers-and-why-do-you-need-them.html>.

TITTEL, E. *Clusters for Dummies*. Nova Jersey, Estados Unidos: John Wiley & Sons, 2010.

TURNBULL, J. *The Docker Book: Containerization is the new virtualization*. Nova York, Estados Unidos: James Turnbull, 2016.

TWISTLOCK. *All About Containers*. 2016. Disponível em: <https://www.twistlock.com/container-whitepaper-chapter-1/>.

YU, Y. *OS-level Virtualization and Its Applications*. Tese (Doutorado) — Stony Brook University, 2007.

ZHOU, Y. et al. *Comparison of Virtualization and Containerization Techniques for High-Performance Computing*. 2015.

ANEXO A – Configuração *Docker*

```
clusterhpc:~ # docker info
Containers: 1
 Running: 1
 Paused: 0
 Stopped: 0
Images: 4
Server Version: 1.12.1
Storage Driver: overlay
 Backing Filesystem: extfs
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: host bridge null overlay
Swarm: active
 NodeID: cu4btmfu4yjqcua3edtc5uune
 Is Manager: true
 ClusterID: 0bipyuw4nwn100feq4gwwp54g
 Managers: 1
 Nodes: 11
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Heartbeat Tick: 1
  Election Tick: 3
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
 Node Address: 172.23.0.1
Runtimes: runc
Default Runtime: runc
Security Options: apparmor seccomp
Kernel Version: 4.1.27-27-default
Operating System: SUSE Linux Enterprise Server
   12
OSType: linux
Architecture: x86_64
CPUs: 8
```

```
Total Memory: 23.55 GiB
Name: clusterhpc
ID: BG55:3OIV:BUJ5:RVQL:RUSC:6ETB:FLDP:MZIB:HZO7
    :UWXZ:ZEG6:N6C7
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
WARNING: No swap limit support
WARNING: No kernel memory limit support
Insecure Registries:
 127.0.0.0/8
```

ANEXO B – *Dockerfile* - EnergyPlus

```
# https://github.com/NREL/docker-energyplus/blob
    /master/Dockerfile

FROM ubuntu:14.04

MAINTAINER Nicholas Long nicholas.long@nrel.gov

# This is not ideal. The tarballs are not named
    nicely and EnergyPlus versioning is strange
ENV ENERGYPLUS_VERSION 8.5.0
ENV ENERGYPLUS_TAG v8.5.0
ENV ENERGYPLUS_SHA c87e61b44b

# This should be x.y.z, but EnergyPlus
    convention is x-y-z
ENV ENERGYPLUS_INSTALL_VERSION 8-5-0

# Downloading from Github
# e.g. https://github.com/NREL/EnergyPlus/
    releases/download/v8.3.0/EnergyPlus-8.3.0-6
    d97d074ea-Linux-x86_64.sh
ENV ENERGYPLUS_DOWNLOAD_BASE_URL https://github.
    com/NREL/EnergyPlus/releases/download/
    $ENERGYPLUS_TAG
ENV ENERGYPLUS_DOWNLOAD_FILENAME EnergyPlus-
    $ENERGYPLUS_VERSION-$ENERGYPLUS_SHA-Linux-
    x86_64.sh
ENV ENERGYPLUS_DOWNLOAD_URL
    $ENERGYPLUS_DOWNLOAD_BASE_URL/
    $ENERGYPLUS_DOWNLOAD_FILENAME


# Collapse the update of packages, download and
    installation into one command
# to make the container smaller & remove a bunch
     of the auxiliary apps/files
# that are not needed in the container
RUN apt-get update && apt-get install -y ca-
    certificates curl \
     && rm -rf /var/lib/apt/lists/* \
```

```
    && curl -SLO $ENERGYPLUS_DOWNLOAD_URL \
    && chmod +x $ENERGYPLUS_DOWNLOAD_FILENAME \
    && echo "y\r" | ./
       $ENERGYPLUS_DOWNLOAD_FILENAME \
    && rm $ENERGYPLUS_DOWNLOAD_FILENAME \
    && cd /usr/local/EnergyPlus-
       $ENERGYPLUS_INSTALL_VERSION \
    && rm -rf DataSets Documentation
       ExampleFiles WeatherData MacroDataSets
       PostProcess/convertESOMTRpgm \
    PostProcess/EP-Compare PreProcess/FMUParser
       PreProcess/ParametricPreProcessor
       PreProcess/IDFVersionUpdater

# Remove the broken symlinks
RUN cd /usr/local/bin \
    && find -L . -type l -delete

# Add in the test files
ADD test /usr/local/EnergyPlus-
   $ENERGYPLUS_INSTALL_VERSION/test_run
RUN cp /usr/local/EnergyPlus-
   $ENERGYPLUS_INSTALL_VERSION/Energy+.idd \
        /usr/local/EnergyPlus-
           $ENERGYPLUS_INSTALL_VERSION/test_run/

VOLUME /var/simdata
WORKDIR /var/simdata

CMD [ "/bin/bash" ]
```

ANEXO C – *Dockerfile* - Gromacs

```
# https://bitbucket.org/fourplusone_uni/docker-
    gromacs/src/

FROM debian:jessie
RUN apt-get update && apt-get install -y wget

RUN wget ftp://ftp.gromacs.org/pub/gromacs/
    gromacs-5.0.4.tar.gz

RUN echo "c177ae5fd6d71e2bec7369bc66cd082e
    gromacs-5.0.4.tar.gz" > MD5SUM
RUN md5sum -c MD5SUM
RUN tar -xf gromacs-5.0.4.tar.gz

RUN apt-get update && apt-get install -y build-
    essential cmake file libxml2-dev libboost-dev

WORKDIR gromacs-5.0.4
RUN mkdir build
WORKDIR build
RUN cmake .. -DGMX_BUILD_OWN_FFTW=ON -
    DREGRESSIONTEST_DOWNLOAD=ON
RUN make -j `nproc`

# Don't run make check, since mdrun won't work
    on dockers build servers
# RUN make check
RUN make install

WORKDIR /root
RUN echo source /usr/local/gromacs/bin/GMXRC >>
    .profile

CMD /bin/bash -l
```

ANEXO D – *Dockerfile* - OpenFOAM

```
# https://github.com/qnib/docker-openfoam/blob
    /12.04_of230/Dockerfile

FROM qnib/u_compute:u12.04
MAINTAINER "Christian Kniep <christian@qnib.org
    >"

RUN echo "deb http://www.openfoam.org/download/
    ubuntu $(lsb_release -cs) main" > /etc/apt/
    sources.list.d/openfoam.list
RUN echo "2014-10-02.1";apt-get update
### openfoam
RUN apt-get install -y libgl1-mesa-glx libgmp10
    libqt4-opengl libqtcore4 libqtgui4
RUN apt-get install -y binutils cpp-4.4 g++-4.4
    gcc-4.4 gcc-4.4-base libstdc++6-4.4-dev
RUN apt-get install -y libibverbs-dev libopenmpi
    -dev openmpi-common  mpi-default-dev
RUN apt-get install -y libc-bin libc-dev-bin
    libc6 libc6-dev libdrm-dev libdrm-nouveau2
    libgl1-mesa-dev libglu1-mesa libglu1-mesa-dev
     \
                         libgmp-dev libgmpxx4ldbl
                            libicu48 libmpfr-dev
                            libmpfr4 libnuma1
                            libpthread-stubs0
                            libpthread-stubs0-dev
                            \
                         libqt4-designer libqt4-
                            dev libqt4-help libqt4
                            -opengl-dev libqt4-
                            qt3support libqt4-
                            scripttools libqt4-svg
                             libqt4-test \
                         libqtwebkit-dev
                            libqtwebkit4
                            libtorque2 \
                         libx11-dev libx11-doc
                            libxau-dev libxcb1-dev
                             libxdmcp-dev libxext-
```

```
                            dev linux-libc-dev
                            manpages manpages-dev
                            \
                        mesa-common-dev qt4-
                            linguist-tools qt4-
                            qmake \
                        x11proto-core-dev
                            x11proto-input-dev
                            x11proto-kb-dev
                            x11proto-xext-dev xorg
                            -sgml-doctools xtrans-
                            dev zlib1g-dev
RUN apt-get install -y libscotch-5.1 libscotch-
    dev
RUN apt-get install -y binutils libboost-date-
    time1.46-dev libboost-date-time1.46.1
    libboost-dev
RUN apt-get install -y libboost-program-options-
    dev libboost-program-options1.46.1 libboost-
    serialization1.46-dev libboost-serialization1
    .46.1
RUN apt-get install -y libboost-thread-dev
    libboost-thread1.46-dev libboost1.46-dev

RUN apt-get install -y build-essential
RUN apt-get install -y libptscotch-dev

ADD dpkg /opt/dpkg/
RUN dpkg -i /opt/dpkg/*
RUN apt-get install -y --force-yes openfoam230

RUN sed -i -e 's/   allowSystemOperations.*/
    allowSystemOperations   1;/' $(find /opt/
    openfoam*/etc -name controlDict|head -n1)
# ENV
RUN echo "source $(find /opt/openfoam*/etc -name
     bashrc|head -n1)" >> /root/.bashrc

CMD supervisord -c /etc/supervisord.conf
```

ANEXO E – Artigo do TCC

LIFTER - Disponibilização de aplicações via *containers* de *software* em um *cluster* de alto desempenho

José Norberto Guiz Fernandes Corrêa[1]

[1]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

Abstract. *With the recent development in software container technologies, the main one being Docker, a more flexible approach to distribute software became possible. The present work employed this technology, through the use of Docker, in a high-performance cluster at SeTIC/UFSC. The goal was the reduction of the time spent deploying applications to this environment, when compared with the previous factory solution. With the deployment of Docker, and using the factory solution data as a baseline, it was possible to obtain near 94% of disk space usage reduction for the application storage, and a range of 78% to 96% reduction in the time spent deploying applications to the cluster.*

Resumo. *Com a modernização de tecnologias de containers de software, em especial do Docker, tornou-se possível uma abordagem mais flexível na disponibilização de aplicações. Através do Docker, este trabalho utilizou-se desta tecnologia em um cluster de alto desempenho da SeTIC/UFSC, de forma a reduzir os recursos e o tempo despendido na disponibilização de aplicações em relação à solução de fábrica utilizada anteriormente neste ambiente. Com a utilização do Docker, e tendo como base a solução de fábrica, foram obtidas reduções próximas de 94% no espaço em disco utilizado para armazenamento das aplicações, e de 78% a 96% no tempo despendido para disponibilização destas aplicações no cluster.*

1. Introdução

Em diversas áreas científicas e profissionais (tais como engenharia, biologia, meteorologia e astronomia, entre outras), é crescente a demanda de recursos computacionais de alto desempenho, uma vez que estes são essenciais na solução de problemas cada vez mais complexos. Porém, devido ao seu alto custo de aquisição e manutenção, tais ambientes computacionais são geralmente compartilhados entre diversos grupos de pesquisa, de modo a reduzir o custo total entre si. Assim, diferentes aplicações, com requisitos e finalidades distintas, passam a coexistir dentro destes ambientes. No entanto, mesmo para um número pequeno de grupos de usuários, surgem problemas oriundos deste compartilhamento de recursos: conflitos de dependências entre aplicações, indisponibilidade de recursos, problemas de usabilidade, entre outros. Com a modernização de tecnologias de *containers* de software, tornou-se possível uma abordagem mais flexível na disponibilização de aplicações.

De forma a tratar os problemas encontrados em um *cluster* de alto desempenho da SeTIC/UFSC, principalmente o conflito de dependências e o elevado tempo despendido na disponibilização de aplicações, este trabalho consistiu na implementação de uma tecnologia de *containers* de *software* neste ambiente.

Os principais autores que embasaram este trabalho foram [Jacobsen and Canon 2015], [Dantas 2005] e [Meffe et al. 2006].

1.1. Objetivos

Este trabalho teve como objetivo geral a configuração e implementação de uma ferramenta de gerenciamento de *containers* de *software* em um ambiente de *cluster* de alto desempenho. A contribuição pretendida deste trabalho é a de reduzir o tempo de disponibilização de aplicações no *cluster* e facilitar a administração deste ambiente, de forma a obter um melhor aproveitamento dos recursos computacionais existentes.

Os objetivos específicos deste trabalho foram:

1. Realização de um estudo de arquiteturas distribuídas;
2. Realização de um estudo de tecnologias de *containers* de *software*;
3. Configuração do *cluster* de alto desempenho para suporte a *containers* de *software*;
4. Disponibilização de aplicações-exemplo;
5. Avaliação dos ambientes prévio e posterior.

1.2. Método de Pesquisa

Este trabalho caracterizou-se como uma pesquisa aplicada, pois objetivou gerar conhecimentos para aplicação prática dirigidos à solução de problemas específicos de um *cluster HPC* da SeTIC/UFSC. As etapas deste trabalho foram divididas em:

- Etapa 1: Análise da fundamentação teórica;
- Etapa 2: Revisão do estado da arte;
- Etapa 3: Configuração do ambiente de *cluster*;
- Etapa 4: Disponibilização de aplicações e coleta de dados;
- Etapa 5: Discussão dos resultados obtidos.

2. Fundamentação Teórica

2.1. Computação Distribuída

Segundo [Dantas 2005], define-se a computação distribuída como um segmento da Ciência da Computação que tem como objetivo a melhoria do desempenho de aplicações distribuídas e paralelas, utilizando-se de complexas infraestruturas computacionais. É um conceito que engloba, de acordo com [Tittel 2010], múltiplos sistemas computacionais trabalhando sobre um determinado problema, sendo este problema dividido em várias partes, cada parte solucionada por computadores distintos, que se comunicam entre si através de uma rede.

Para [Coulouris et al. 2005], os sistemas distribuídos, sob a visão da arquitetura de máquinas, são considerados configurações com grande capacidade de escalabilidade, pela possibilidade de agregação dos computadores existentes nas redes convencionais em um sistema único, onde a homogeneidade ou heterogeneidade do conjunto de máquinas permite a formação de diferentes configurações.

2.1.1. *Clusters* e Processamento de Alto Desempenho

[Tittel 2010] define *cluster* de computadores como um agrupamento interconectado de nós computacionais, operando colaborativamente na execução de aplicações especiais e coordenados por um *software* gerenciador. Além disso, outras razões são consideradas por organizações no uso destes tipos de estruturas computacionais, tais como:

- Redução no tempo para a solução, principalmente em simulações, modelos ou previsões de larga escala e complexas;
- Aproveitamento máximo do investimento já feito em recursos computacionais;
- Estabelecimento de controle das tarefas (também chamadas de *jobs*) e priorização de trabalhos mais importantes para um processamento mais ágil;
- Melhoria geral da confiança, manutenção e disponibilidade computacional por um custo reduzido.

O processamento de alto desempenho (*HPC - High-Performance Computing*) é descrito por [Eadline 2009] como uma classe de supercomputadores e *clusters* que solucionam problemas avançados e de alta intensidade computacional. Com o objetivo de aumentar a vazão computacional, um *cluster HPC* é, geralmente, utilizado de duas maneiras:

- Executando concorrentemente diversas tarefas, conforme a Figura 1. É recomendado quando se necessita rodar *jobs* idênticos com parâmetros ou conjuntos de dados distintos. Dependendo dos recursos computacionais existentes, os *jobs* podem precisar aguardar em uma fila o término daqueles que estão em execução. Neste modo, o processamento de um *job* é feito localmente em um nó do *cluster*, ou seja, não existe comunicação com outros nós; além disso, como os *jobs* são independentes, também não existe interação entre eles.

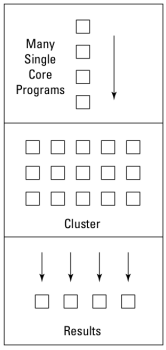

Figura 1. Processamento concorrente de *jobs* em um *cluster* [Eadline 2009]

- Dividindo um grande *job* em subtarefas e executar cada subtarefa em nós distintos do *cluster*, conforme a Figura 2. Este procedimento é realizado a nível de *software*

e, portanto, as aplicações necessitam ser modificadas para se utilizarem desta funcionalidade. O método mais comum para desenvolvimento de programas paralelos é a utilização de troca de mensagens com a biblioteca MPI (*Message Passing Interface*). Como as subtarefas comunicam-se entre si, o tráfego de rede pode ser intenso, de modo que redes de alto desempenho são necessárias para lidar com o alto fluxo de dados.

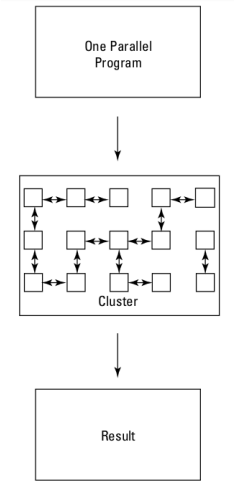

Figura 2. Processamento paralelo de um *job* em um *cluster* [Eadline 2009]

O gerenciamento técnico de *clusters*, segundo [Meffe et al. 2006], envolve as tarefas básicas de instalação, configuração, manutenção, monitoramento e escalonamento de recursos. Um ambiente compartilhado por múltiplos projetos ou grupos de usuários necessita de priorização e alocação de recursos eficientes, de forma a garantir resultados para todos os envolvidos. De acordo com [Eadline 2009], quanto mais um negócio ou organização depende dos seus ambientes de *clusters* para finalidades de processamento, produção ou receita, mais essencial torna-se a gerência técnica.

Para [Tittel 2010], a usabilidade de um *cluster* para os usuários consiste basicamente das aplicações que este suporta, de modo que a facilidade em disponibilizar estas aplicações agiliza o atendimento às demandas dos mesmos. As aplicações utilizadas atualmente em *HPC* abrangem vários setores, tais como:

- Engenharias: design 2D e 3D, modelagem, simulações, verificações;
- Economia e finanças: análises de risco, predições;
- Geociências: exploração de gás e petróleo;
- Meteorologia: modelagem climática, previsão;
- Biociências: detecção e prevenção de doenças, desenvolvimento de medicamentos;
- Criação de conteúdo digital: processamento digital, renderização;

- Universidades e governos: pesquisa básica e aplicada.

A lista torna-se cada vez maior, pois mais áreas descobrem no processamento de alto desempenho uma ferramenta extremamente útil para a melhor compreensão das suas ciências, mercados e produtos.

2.2. Containers de Software

De acordo com [Yu 2007], a virtualização a nível de sistema operacional é definida como a existência de múltiplas instâncias isoladas de espaços de usuário (ao invés de apenas uma), que são gerenciadas pelo *kernel* deste sistema. Estas instâncias, chamadas de *containers* de software (ou simplesmente *containers*), representam todo um ambiente de execução: uma aplicação, juntamente com todas suas dependências, bibliotecas e arquivos de configuração, agrupados em um único "pacote". Ao criar um *container* de uma determinada aplicação e suas dependências, a diversidade de sistemas operacionais e infraestruturas subjacentes são abstraídas.

Máquinas virtuais e *containers* possuem funcionalidades similares quanto ao isolamento e alocação de recursos computacionais. No entanto, uma abordagem arquitetural distinta, ilustrada na Figura 3, permite que os *containers* possuam uma maior portabilidade e eficiência.

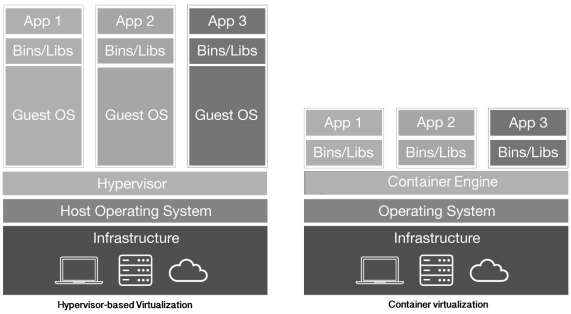

Figura 3. Diagrama estrutural de máquinas virtuais e *containers* [Twistlock 2016]

Na ilustração anterior, três aplicações em *containers* rodam em um único sistema operacional e compartilham o *kernel* do sistema entre si. Isso faz com que os *containers* sejam mais leves e consumam menos recursos do que máquinas virtuais. A parte compartilhada é somente leitura, enquanto que cada *container* possui o seu espaço para escrita. No entanto, conforme [Rubens 2015], *containers* e máquinas virtuais podem ser vistas entre si como tecnologias complementares ao invés de competitivas. Isso se deve ao fato de ser possível rodar *containers* em máquinas virtuais, facilitando, através da virtualização de *hardware*, a gerência da infraestrutura (tais como rede e armazenamento), além de aumentar o isolamento e a segurança.

As raízes da tecnologia de *containers* de *software* datam do final da década de 1970. De acordo com [Gunaratne 2016], os principais marcos no histórico desta tecnologia são:

- ***chroot*:** A ideia por trás dos *containers* iniciou em 1979 com o chroot (*change root*) do UNIX. É uma chamada do sistema operacional UNIX que altera o diretório raiz de um processo e seus processos-filho para um novo local do sistema de arquivos, visível apenas para este processo. Esta funcionalidade fornece um espaço de disco isolado para cada processo.
- ***FreeBSD Jails*:** Desenvolvido em 2000, é uma das primeiras tecnologias de *containers* de *software* existentes. É uma chamada de sistema similar ao chroot, porém com funcionalidades adicionais de proteção de processos, tais como isolamento de sistema de arquivos, usuários e rede. Como resultado, é possível associar um endereço IP para cada *jail* e personalizar instalações de *software* e configurações.
- ***Linux VServer*:** Desenvolvido em 2001, é um mecanismo que pode ser utilizado para particionamento de recursos (sistema de arquivos, tempo de *CPU*, endereços de rede e memória).
- ***Solaris Containers*:** Desenvolvido em 2004, é uma combinação de controladores de recursos de sistema e separação por zonas. As zonas comportam-se como servidores virtuais completamente isolados, dentro de uma única instância do sistema operacional.
- ***OpenVZ*:** Desenvolvido em 2005, faz uso de um *kernel Linux* modificado com a finalidade de fornecer virtualização, isolamento, gerenciamento de recursos e pontos de verificação.
- ***Process Container/cgroups*:** Desenvolvido em 2006 pela Google, possibilita a limitação e isolamento de recursos de um conjunto de processos. Foi renomeado posteriormente para *Control Groups* (cgroups).
- ***LXC*:** Desenvolvido em 2008, é a primeira e mais completa implementação de um gerenciador de *containers Linux*. Tendo como base o uso do cgroups e *namespaces* do *Linux*, é disponibilizado na biblioteca liblxc e possui APIs para diversas linguagens de programação. Em comparação com outras tecnologias de *containers*, o LXC funciona nativamente com o *kernel Linux*.
- ***LMCTFY* (*Let Me Contain That For You*):** Desenvolvido em 2013, é a versão de código-aberto da implementação de *containers Linux* da *Google*, visando garantia de desempenho e *containers* com baixo *overhead*. Em 2015, a *Google* migrou conceitos e abstrações do *LMCTFY* para o projeto *libcontainer*, e por isso não é mais mantido.
- ***Docker*:** Desenvolvido em 2013, é um projeto de código aberto que automatiza a disponibilização de aplicações dentro de *containers* de software. Ele inclui um modelo de imagens em camadas eficiente, repositórios de imagens, interface *CLI*, entre outras funcionalidades. Informações mais detalhadas são apresentadas na seção 2.2.1.
- ***rkt (Rocket)*:**

  Desenvolvido em 2014, é uma iniciativa desenvolvida pelo *CoreOS*, motivada a fornecer uma segurança mais rigorosa para *containers* em ambientes de produção.
- ***Windows Containers*:** Desenvolvido em 2016, é uma iniciativa da *Microsoft* para adicionar o suporte a *containers* de aplicações *Windows* ao sistema operacional *Windows Server 2016*.

2.2.1. Docker

O *Docker* é uma plataforma de disponibilização de aplicações que utiliza funcionalidades de isolamento de recursos do *kernel Linux* para a criação de *containers* independentes. O suporte a *namespaces* do *kernel* isola a visão de uma aplicação em relação ao sistema operacional, incluindo árvore de processos, rede, UIDs e sistemas de arquivos, enquanto que o *cgroups* fornece limitação de recursos, tais como *CPU*, memória, *I/O* e rede. A sua estrutura básica é ilustrada na Figura 4.

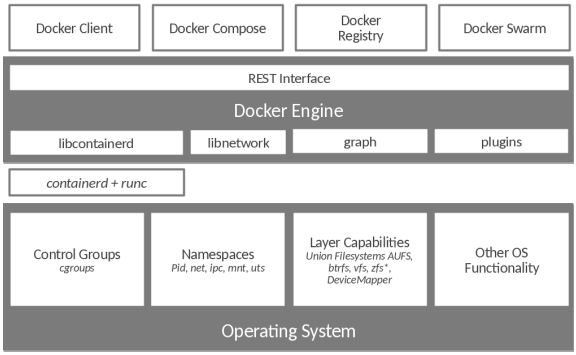

Figura 4. Visão estrutural do *Docker* [Brown and Stark 2016]

Os componentes principais da plataforma *Docker*, de acordo com [Jernigan 2016] e [Turnbull 2016], são:

- ***Docker Engine*:** O *daemon* que gerencia imagens, *containers*, volumes, redes, entre outros. É construído sobre dois elementos principais:
  - ***runC*:** é uma ferramenta *CLI* que interage com as funcionalidades do *kernel Linux* com o objetivo de instanciar e rodar *containers*.
  - ***containerd*:** é um *daemon* que utiliza o *runC* para gerenciar *containers* e expor suas funcionalidades através de uma *API*.
- ***Docker Client*:** É o componente que interage com o *daemon* do *Docker Engine*.
- ***Dockerfile*:** É um arquivo de texto simples, composto por uma sequência de comandos, que funciona como uma "receita" para a criação de *Docker Images*.
- ***Docker Images*:** A base da criação de *containers*, uma imagem é composta da união de camadas de sistemas de arquivos, empilhadas umas sobre as outras, conforme a Figura 5. Uma imagem não possui estado e é imutável.
- ***Docker Registry*:** São repositórios de imagens.
- ***Docker Containers*:** Os *containers* propriamente ditos, instâncias criadas a partir de imagens.
- ***Docker Compose*:** Possibilita a criação conjunta de grupos de *containers* para um propósito comum.
- ***Docker Swarm*:** Possibilita a orquestração de *clusters* de *containers*, permitindo a execução escalável de aplicações.

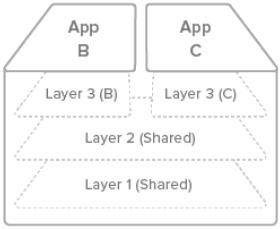

Figura 5. Diagrama de camadas de uma imagem *Docker* [Ellingwood 2015]

Tanto o *Docker* quanto as demais soluções existentes possuem limitações quanto à interoperabilidade entre diferentes arquiteturas de processadores e sistemas operacionais: imagens criadas para uma arquitetura *x86-64* não funcionarão nativamente para arquiteturas *ARM* ou *ARM64*, por exemplo, assim como imagens criadas para um sistema operacional *Linux* não funcionarão nativamente para sistemas operacionais *Windows*. É possível o uso de emuladores (como o *QEMU*) ou tradutores (como o *Wine*), de modo a lidar com estas limitações, porém o seu desempenho não é garantido para todos os casos.

3. Trabalhos Relacionados

Como suporte para a realização deste trabalho, foi realizada uma pesquisa de produções relevantes nas bases *IEEE Xplore Digital Library*, *ACM Digital Library* e *Google Scholar* envolvendo os termos "*container*", "*cluster*" e "*HPC*". Desta busca, foram selecionadas obras apresentadas na Tabela 1. Na seção a seguir, são comentados alguns dos resultados relevantes destas obras para o desenvolvimento deste trabalho.

O trabalho de [Jacobsen and Canon 2015] relatou a implementação da infraestrutura de *HPC* baseada em *containers* no *NERSC* (*National Energy Research Scientific Computing Center*), um centro gerenciado pelo *Lawrence Berkeley National Laboratory* e pelo *Department of Energy Office of Science*. O protótipo desenvolvido pelo *NERSC*, chamado *Shifter*, foi uma maneira escalável de disponibilizar *containers* em um ambiente de *HPC*. O protótipo converte imagens *Docker* e máquinas virtuais em um formato comum, e este formato permite a distribuição das imagens nos supercomputadores *Cray* da organização. A interface do *Shifter* permite ao usuário selecionar uma imagem de um repositório privado ou do *Docker Hub* e, posteriormente, submeter tarefas ao ambiente, que são executadas inteiramente dentro de *containers*. O fluxo de trabalho deste protótipo é ilustrado na Figura 6.

O objetivo do trabalho foi solucionar a rápida expansão das aplicações, bibliotecas e ferramentas demandadas pelas comunidades de usuários deste ambiente. Em diversos casos, os requisitos de duas comunidades distintas conflitavam (distribuições de *Linux* distintas, versões diferentes de compiladores, etc). As principais necessidades dos usuários consistiam em poder executar suas aplicações científicas no mesmo contexto em que eram desenvolvidas e, em alguns casos, poder facilmente migrar do seu ambiente *desktop* para o ambiente de *HPC*. Algumas comunidades tentaram buscar na computação em nuvem esta flexibilidade, porém encontraram desafios, tais como gerenciar sistemas de arquivos e provisionar recursos, que são normalmente resolvidos por um *cluster* gerenciado. A

Tabela 1. Trabalhos selecionados

Título	Autoria
Contain this, unleashing Docker for HPC	[Jacobsen and Canon 2015]
Is container-based technology a winner for high performance scientific applications?	[Adufu et al. 2015]
How advanced cloud technologies can impact and change HPC environments for simulation	[Mancini and Aloisio 2015]
An updated performance comparison of virtual machines and Linux containers	[Felter et al. 2015]
Orchestrating Docker Containers in the HPC Environment	[Higgins et al. 2015]
HPC Made Easy: Using Docker to Distribute and Test Trilinos	[Deal 2016]
An introduction to Docker for reproducible research	[Boettiger 2015]
Comparison of Virtualization and Containerization Techniques for High-Performance Computing	[Zhou et al. 2015]

utilização de *containers* neste contexto permitiu a flexibilidade dos sistemas em nuvem, juntamente com um desempenho próximo do *hardware* nativo.

A implementação de um ambiente de *HPC* com suporte a *containers* de *software* trouxe flexibilidade e facilidade de uso para os seus usuários, com praticamente nenhuma degradação de *performance* em relação ao desempenho nativo na maioria dos casos. Ainda que a tecnologia de *containers* seja relativamente recente nas comunidades científicas e de *HPC*, as vantagens apresentadas podem fazer com que sua adoção seja acelerada nos próximos anos.

No trabalho de [Adufu et al. 2015], a alocação eficiente de recursos, o tempo de execução de aplicações e a demora em disponibilizar novas aplicações em ambientes de *HPC* foram as dificuldades identificadas. As autoras investigaram o uso de *containers Docker* na execução de uma aplicação científica de alto desempenho (*Autodock3*, um software de modelagem molecular), comparando o desempenho com máquinas virtuais (*VMs*).

Os resultados demonstraram que os tempos de execução utilizando *containers* são menores quando comparados com máquinas virtuais, conforme a Figura 7, principal-

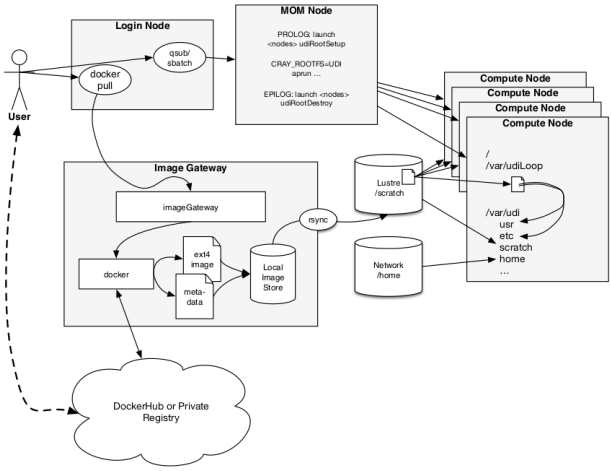

Figura 6. Fluxo de trabalho do *Shifter* [Jacobsen and Canon 2015]

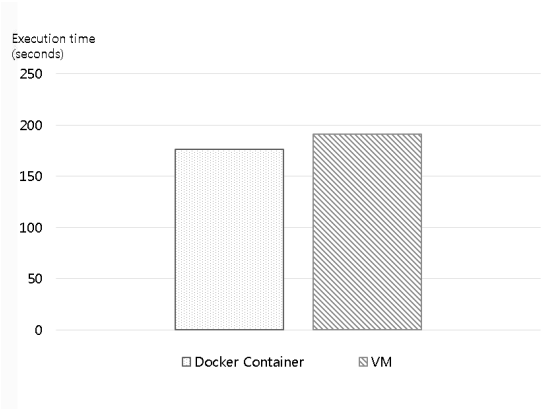

Figura 7. Comparativo entre *containers* e *VMs* [Adufu et al. 2015]

mente nos tempos de inicialização. Os experimentos também demonstram que o *Docker* gerencia os recursos de memória mais eficientemente quando quantidades maiores que a memória física são alocadas para as instâncias em execução, conforme a Figura 8.

O trabalho de [Deal 2016] utilizou o *Docker* para avaliar o desempenho paralelo de testes utilizando o projeto *Trilinos*, um *framework* de *software* orientado a objetos que visa o desenvolvimento de algoritmos e tecnologias para solucionar problemas científicos e de engenharia. Os resultados da execução do *Epetra BasicPerfTest* em um *cluster* e em *containers* mostraram que a diferença de *performance* é praticamente inexistente. Ao

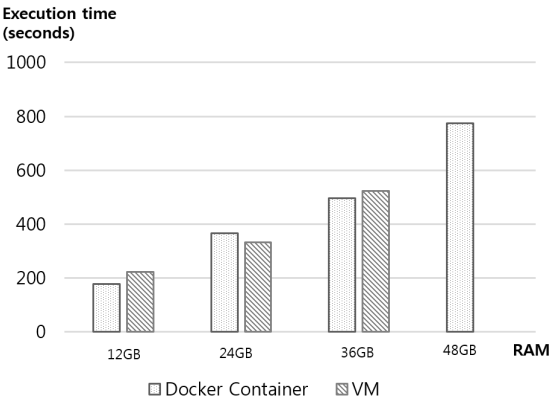

Figura 8. Comparativo entre diferentes configurações de memória [Adufu et al. 2015]

executar em paralelo com *MPI*, os resultados foram mais irregulares, apesar de os *containers* terem um desempenho consistente, próximo ao *bare-metal*. Para tarefas com um maior número de processos (a partir de 48), os *containers* superaram o desempenho nativo, o que sugere que o uso de *containers* consegue otimizar a execução de aplicações escaláveis.

Dentre os trabalhos selecionados, o produzido por [Jacobsen and Canon 2015] foi o que mais se aproximou das dificuldades encontradas no *cluster HPC* da SeTIC/UFSC (apresentado no capítulo a seguir) e dos resultados práticos almejados para este ambiente. Resultados similares também foram apresentados no trabalho de [Higgins et al. 2015].

Já os trabalhos de [Adufu et al. 2015], [Deal 2016], [Zhou et al. 2015] e [Felter et al. 2015] compararam o desempenho de aplicações em diferentes estruturas (*containers*, *bare-metal* e máquinas virtuais), onde os resultados dos mesmos justificam, de maneira quantitativa, o uso de *containers* para aplicações científicas.

O trabalho de [Boettiger 2015] destaca a vantagem no uso do *Docker* em vários ambientes computacionais, principalmente como uma forma de facilitar a reprodução de trabalhos científicos. Como estes ambientes computacionais possuem naturalmente uma evolução rápida, a repetição e extensão de experimentos diversos torna-se um grande desafio.

Por fim, o trabalho de [Mancini and Aloisio 2015] descreve como soluções de virtualização e *cloud computing*, adotadas por diversas empresas e organizações nos últimos anos, vêm sendo também adotadas pela área de *HPC*.

Todos as obras mencionadas anteriormente foram fundamentais para o desenvolvimento deste trabalho, pois tornaram plausíveis a implementação e a utilização de *containers* em um ambiente de *cluster HPC*. De um modo geral, estas obras embasaram e justificaram a realização do Projeto *Lifter*, apresentado no próximo capítulo.

4. Projeto *Lifter*

4.1. Ambiente de trabalho

Este trabalho foi realizado em um *cluster* de alto desempenho da SeTIC/UFSC, composto por um servidor *SGI Altix XE250* (*head node*), um servidor *SGI Altix XE210*, quatro servidores *SGI Altix XE310* (dois nós de processamento por *blade*) e um servidor *SGI Altix XE320* (dois nós de processamento por *blade*), interconectados por interfaces de rede *Gigabit Ethernet* através de dois *switches* dedicados, conforme ilustrado na Figura 9. As especificações do *cluster* são apresentadas na Tabela 2.

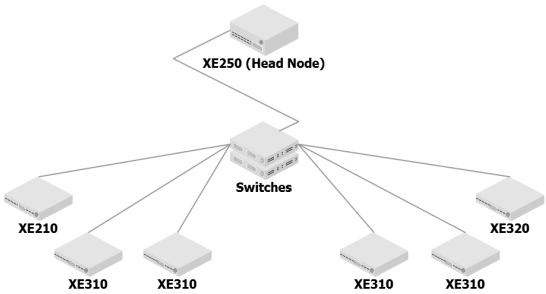

Figura 9. *Cluster HPC* SeTIC/UFSC

Tabela 2. Especificações do *cluster HPC* SeTIC/UFSC

Servidor	Nome	*CPU*	*RAM*	Disco	Sistema Operacional	*Kernel*
XE250	admin	2 x Intel Xeon E5420	24GB	1TB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n001	2 x Intel Xeon E5420	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n002	2 x Intel Xeon E5420	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n003	2 x Intel Xeon E5420	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n004	2 x Intel Xeon E5420	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n005	2 x Intel Xeon E5345	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n006	2 x Intel Xeon E5345	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE320	n007	2 x Intel Xeon E5462	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE320	n008	2 x Intel Xeon E5462	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n009	2 x Intel Xeon E5345	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE310	n010	2 x Intel Xeon E5345	16GB	232GB	*SUSE Linux Enterprise 12*	4.1.27-27
XE210	n011	2 x Intel Xeon E5140	16GB	1.5TB	*SUSE Linux Enterprise 12*	4.1.27-27

O *cluster* é atualmente utilizado por um número muito reduzido de grupos de pesquisa da UFSC, justamente pela dificuldade em agregar demandas distintas neste mesmo ambiente. Para remediar a situação, a resolução de conflitos entre aplicações deste ambiente era tratada com uma solução da própria fabricante do *cluster* (*SGI*), consistindo de diversos *scripts* (*cinstallman*, *cpower* e outros), que automatizavam a criação de imagens completas (*dumps*) de discos e disponibilizavam estas imagens para os nós de processamento via *boot* PXE (*Preboot eXecution Environment*). Apesar de solucionar o problema inicial, outros tornaram-se aparentes:

- Necessidade de reiniciar nós do *cluster* para utilização de aplicações distintas;
- Demora no *boot* via *PXE* nos nós do *cluster*, tempo em que os recursos ficam indisponíveis;
- Consumo de espaço em disco elevado para armazenamento das imagens.

4.2. Proposta

O projeto *Lifter* foi desenvolvido tendo em vista os problemas identificados anteriormente e tendo como base os trabalhos relacionados encontrados. Ele propõe o uso de containers de modo a reduzir o tempo para disponibilização, reduzir o uso de espaço em disco e garantir o isolamento das aplicações no cluster HPC de estudo, possibilitando o seu uso por um número maior de usuários da comunidade acadêmica da UFSC.

No âmbito deste trabalho, a ferramenta gerenciadora de containers utilizada foi o Docker, por ser um software livre (licença Apache 2.0), possuir maior estabilidade e documentação de qualidade, liderar os esforços na criação de um padrão aberto de containers e ter uma comunidade ativa de usuários e desenvolvedores. O Rocket também foi considerado, e apesar dos avanços recentes em seu desenvolvimento, ainda é um projeto que carece de maturidade para uso em um ambiente de produção.

O *Lifter* consistiu na instalação e configuração do Docker em todos os nós do cluster, na criação de um repositório de imagens no *head node* e na disponibilização destas imagens para os demais nós. Nesta estrutura, apresentada na Figura 10, o próprio repositório de imagens é disponibilizado dentro de um container rodando no *head node*. A transferência de uma determinada imagem é realizada apenas uma vez para cada nó, e esta imagem é armazenada localmente no mesmo. Uma vez transferidas as imagens, é possível instanciar em cada nó os containers destas aplicações.

Para efeito de comparação pré e pós-projeto, as aplicações selecionadas para testes, já utilizadas no ambiente de trabalho, foram as seguintes:

- **EnergyPlus:** É um programa de código aberto (licença BSD-3-like), desenvolvido pelo Departamento de Energia dos Estados Unidos, voltado para simulações térmicas e elétricas, utilizado por engenheiros, arquitetos e pesquisadores para auxiliar na modelagem de processos de aquecimento, resfriamento, ventilação e iluminação de edifícios.
- **Gromacs *(GROningen MAchine for Chemical Simulations)*:** É um pacote de código aberto (licença LGPL), desenvolvido pela Universidade de Groningen (Holanda), dedicado a cálculos de dinâmica molecular, tais como simulações de proteínas, lipídios, ácidos nucleicos e polímeros.
- **OpenFOAM:** É um programa de código aberto (licença GPL), desenvolvido pela OpenCFD Ltd. e OpenFOAM Foundation, dotado de ferramentas para análise numérica e voltada para solução de problemas complexos relacionados a dinâmica de fluidos envolvendo reações químicas, turbulência, transferência de calor, acústica, mecânica dos sólidos e eletromagnetismo.

4.3. Implementação

Visando a obtenção de resultados mais fidedignos, foi utilizada a versão mais recente do *Docker* (1.12.1) na época em que o trabalho foi realizado. Foi necessária a instalação manual desta versão em todos os nós do *cluster*, pois a versão presente nos repositórios

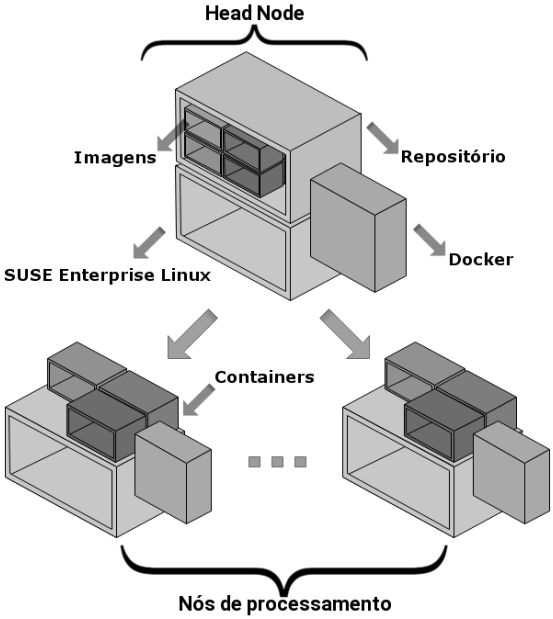

Figura 10. Estrutura do *Lifter*

oficiais era obsoleta. Também foram atualizados os *kernels* dos nós para uma versão estável e recente (*Linux* 4.1.27). Informações adicionais quanto as configurações do Docker podem ser visualizadas no excerto de linha de comando do Anexo 5.

Para a criação das imagens das aplicações, foram utilizados *Dockerfiles* disponíveis no *Docker Hub*, apresentados nos Anexos 5, 5 e 5. As imagens Docker das aplicações foram compiladas e armazenadas em um *Docker Registry* no *head node* do *cluster*. Este repositório roda também dentro de um *container* e é acessível diretamente pelos nós de processamento através da porta 5000 (*TCP*). A decisão em utilizar um repositório privado surgiu pela necessidade de ter um maior controle sobre o local onde as imagens são armazenadas: apesar das aplicações utilizadas como exemplo neste trabalho serem de licença aberta, algumas aplicações proprietárias que possam vir a ser utilizadas no *cluster* (tais como *MATLAB* ou *ANSYS*) não podem ser disponibilizadas em um repositório público como o *Docker Hub*.

4.4. Cenários de teste e resultados

Uma primeira avaliação é a comparação do espaço ocupado em disco das imagens (*dumps*) previamente utilizadas no *cluster* com as imagens *Docker*, apresentadas resumidamente na Tabela 3.

Conforme a Tabela 3, o tamanho das imagens *Docker* representam uma redução de

Tabela 3. Espaço em disco ocupado por imagens

Aplicação	Imagens/*Dumps* de Disco	Imagens *Docker*	Diferença
EnergyPlus 8.5.0	14637MB	268MB	-98.17%
GROMACS 5.0.4	14935MB	976MB	-93.37%
OpenFOAM 2.3.0	16296MB	1525MB	-90.64%
Total:	45868MB	2769MB	-93.96%

espaço em disco próxima de 94% em relação à solução da fabricante, que utiliza *dumps*.

A segunda avaliação consistiu no tempo necessário para a disponibilização das imagens. Foram consideradas duas situações, de modo a se obter um limite mínimo e máximo:

- Tempo de disponibilização para apenas um nó do *cluster*;
- Tempo de disponibilização para todos os nós do *cluster*, paralelamente.

Para cada uma das situações, foram realizadas três medidas e calculada a média. Os resultados obtidos são apresentados nas Tabelas 4 e 5.

Tabela 4. Tempo para disponibilização de imagens em um nó

Aplicação	Imagens/*Dumps* de Disco	Imagens *Docker*	Diferença
EnergyPlus 8.5.0	1280s (21min 20s)	41s	-96.80%
GROMACS 5.0.4	1351s (22min 31s)	145s (2min 25s)	-89.27%
OpenFOAM 2.3.0	1442s (24min 2s)	315s (5min 15s)	-78.16%

Tabela 5. Tempo para disponibilização de imagens no *cluster*

Aplicação	Imagens/*Dumps* de Disco	Imagens *Docker*	Diferença
EnergyPlus 8.5.0	1998s (33min 18s)	77s (1min 17s)	-96.15%
GROMACS 5.0.4	2051s (34min 11s)	218s (3min 38s)	-89.37%
OpenFOAM 2.3.0	2275s (37min 55s)	473s (7min 53s)	-79.21%

Conforme as Tabelas 4 e 5, obteve-se uma redução na faixa de 78% a 96% no tempo de disponibilização utilizando *containers*. Vale ressaltar também o fato de que, além do tempo reduzido, os nós permaneceram ativos e disponíveis para execução de jobs; o que não ocorre com a solução de fábrica do *cluster*, que faz com que os nós fiquem indisponíveis de 21 a 38 minutos enquanto os servidores são reiniciados.

4.5. Considerações

De acordo com os resultados obtidos, é notável que a solução de *containers*, adotada no *Lifter*, é uma alternativa muito mais ágil e leve do que a solução de fábrica do *cluster HPC* da SeTIC/UFSC. Esta abordagem, em contraste com a solução antiga, possibilita:

- A inclusão de mais aplicações no *cluster* e, consequentemente, a possibilidade de expandir a quantidade de usuários que utilizam o ambiente;
- O aumento da disponibilidade dos recursos computacionais para execução de *jobs*;
- O encapsulamento das aplicações e a mobilidade entre ambientes computacionais distintos ao delegar aos usuário a criação das imagens das aplicações.

Os resultados significativos, obtidos com o uso de imagens *Docker*, é devido ao fato destas imagens armazenarem apenas do conteúdo relacionado à própria aplicação, enquanto que as imagens da solução de fábrica armazenam, além da aplicação, todo o sistema operacional. Isso acaba influenciando tanto no tamanho das imagens quanto no tempo necessário para transferência das imagens aos nós do *cluster*.

5. Conclusão

Com o aumento da necessidade por recursos computacionais de alto desempenho, as soluções já consolidadas podem não ser mais suficientes para se atender a demanda. A velocidade das pesquisas e negócios é cada vez maior, e não há espaço para ineficiências nos recursos utilizados. A adoção de tecnologias modernas de *containers* de software em ambientes de *HPC*, conforme este trabalho buscou mostrar, pode contribuir na solução de algumas destas dificuldades.

O trabalho conseguiu atingir o seu objetivo em reduzir o tempo dispendido na disponibilização de aplicações em um *cluster* de alto desempenho da SeTIC/UFSC, além de reduzir a utilização de espaço em disco das imagens de aplicações em relação à solução utilizada anteriormente. Não foram realizados testes intensivos de modo a verificar a diferença de desempenho entre os ambientes pré e pós-*containers*, pois outros trabalhos relacionados já investigaram esta questão em ambientes similares. As ferramentas utilizadas neste trabalho são baseadas na experiência do autor em áreas paralelas (como desenvolvimento de software e gerência de infraestrutura de TI), e representam apenas uma fração das opções disponíveis atualmente.

Algumas das principais dificuldades encontradas na realização deste trabalho foram devidas ao próprio hardware legado do *cluster*, datado de 2008. Os problemas concentraram-se na falha de alguns componentes (como fontes de alimentação e placas de rede), além da utilização de redes *Gigabit Ethernet*, consideradas de baixo desempenho para aplicações *HPC* modernas. No entanto, com o desenvolvimento deste e de trabalhos futuros, espera-se garantir uma sobrevida para este ambiente, que ainda possui um poder de processamento considerável e suficiente para atender várias demandas da comunidade acadêmica da UFSC.

Trabalhos futuros poderão ser realizados com o uso de ferramentas alternativas (como o Rocket), otimizações na configuração do *kernel Linux* e gerenciador de *containers*, e o desenvolvimento de uma ferramenta de gerenciamento de *containers* e imagens de aplicações no *cluster* via interface Web, por exemplo, visando facilitar a interação dos usuários com o ambiente e a execução de aplicações e, consequentemente, reduzir o tempo ocioso da estrutura existente.

Referências

Adufu, T., Choi, J., and Kim, Y. (2015). Is container-based technology a winner for high performance scientific applications? In *Network Operations and Management*

Symposium (APNOMS), 2015 17th Asia-Pacific, pages 507–510, Busan, Coréia do Sul. IEEE.

Boettiger, C. (2015). An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79.

Brown, T. and Stark, J. (2016). Windows server and docker - the internals behind bringing docker and containers to windows. Disponível em: ¡http://www.slideshare.net/Docker/windows-server-and-docker-the-internals-behind-bringing-docker-and-containers-to-windows-by-taylor-brown-and-john-starks/¿.

Coulouris, G. F., Dollimore, J., and Kindberg, T. (2005). *Distributed systems: concepts and design*. Pearson Education, Londres, Reino Unido.

Dantas, M. A. R. (2005). *Computação distribuída de alto desempenho: redes, clusters e grids computacionais*. Axcel Books, Rio de Janeiro, Brasil.

Deal, S. J. (2016). Hpc made easy: Using docker to distribute and test trilinos.

Eadline, D. (2009). *High Performance Computing for Dummies*. Wiley, Indianápolis, Estados Unidos.

Ellingwood, J. (2015). The docker ecosystem: An introduction to common components. Disponível em: ¡https://www.digitalocean.com/community/tutorials/the-docker-ecosystem-an-introduction-to-common-components/¿.

Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. (2015). An updated performance comparison of virtual machines and linux containers. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*, pages 171–172, Filadélfia, Estados Unidos. IEEE.

Gunaratne, I. (2016). Evolution of linux containers and future. Disponível em: ¡http://imesh.io/evolution-of-linux-containers-and-future/¿.

Higgins, J., Holmes, V., and Venters, C. (2015). Orchestrating docker containers in the hpc environment. In *International Conference on High Performance Computing*, pages 506–513, Bengaluru, Índia. Springer.

Jacobsen, D. M. and Canon, R. S. (2015). Contain this, unleashing docker for hpc. *Proceedings of the Cray User Group*.

Jernigan, T. (2016). Docker 1.11 et plus: Engine is now built on runc and containerd. Disponível em: ¡https://medium.com/@tiffanyfayj/docker-1-11-et-plus-engine-is-now-built-on-runc-and-containerd-a6d06d7e80ef¿.

Mancini, M. and Aloisio, G. (2015). How advanced cloud technologies can impact and change hpc environments for simulation. In *High Performance Computing & Simulation (HPCS), 2015 International Conference on*, pages 667–668, Amsterdã, Holanda. IEEE.

Meffe, C., Mussi, E. O. d. P., and Mello, L. R. d. (2006). *Guia de Estruturação e Administração do Ambiente de Cluster e Grid*. Secretaria de Logística e Tecnologia da Informação, Brasília, Brasil, 1ª edition.

Rubens, P. (2015). What are containers and why do you need them? Disponível em: ¡http://www.cio.com/article/2924995/enterprise-software/what-are-containers-and-why-do-you-need-them.html¿.

Tittel, E. (2010). *Clusters for Dummies*. John Wiley & Sons, Nova Jersey, Estados Unidos.

Turnbull, J. (2016). *The Docker Book: Containerization is the new virtualization*. James Turnbull, Nova York, Estados Unidos.

Twistlock (2016). All about containers. Disponível em: ¡https://www.twistlock.com/container-whitepaper-chapter-1/¿.

Yu, Y. (2007). *OS-level Virtualization and Its Applications*. PhD thesis, Stony Brook University.

Zhou, Y., Subramaniam, B., Keahey, K., and Lange, J. (2015). Comparison of virtualization and containerization techniques for high-performance computing.

Anexo A – Configuração Docker

```
clusterhpc:~ # docker info
Containers: 1
 Running: 1
 Paused: 0
 Stopped: 0
Images: 4
Server Version: 1.12.1
Storage Driver: overlay
 Backing Filesystem: extfs
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: host bridge null overlay
Swarm: active
 NodeID: cu4btmfu4yjqcua3edtc5uune
 Is Manager: true
 ClusterID: 0bipyuw4nwn100feq4gwwp54g
 Managers: 1
 Nodes: 11
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Heartbeat Tick: 1
  Election Tick: 3
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
```

```
  Expiry Duration: 3 months
 Node Address: 172.23.0.1
Runtimes: runc
Default Runtime: runc
Security Options: apparmor seccomp
Kernel Version: 4.1.27-27-default
Operating System: SUSE Linux Enterprise Server 12
OSType: linux
Architecture: x86_64
CPUs: 8
Total Memory: 23.55 GiB
Name: clusterhpc
ID: BG55:3OIV:BUJ5:RVQL:RUSC:6ETB:FLDP:MZIB:HZO7:UWXZ:ZEG6:
   N6C7
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
WARNING: No swap limit support
WARNING: No kernel memory limit support
Insecure Registries:
 127.0.0.0/8
```

Anexo B – Dockerfile – EnergyPlus

```
# https://github.com/NREL/docker-energyplus/blob/master/
   Dockerfile

FROM ubuntu:14.04

MAINTAINER Nicholas Long nicholas.long@nrel.gov

# This is not ideal. The tarballs are not named nicely and
   EnergyPlus versioning is strange
ENV ENERGYPLUS_VERSION 8.5.0
ENV ENERGYPLUS_TAG v8.5.0
ENV ENERGYPLUS_SHA c87e61b44b

# This should be x.y.z, but EnergyPlus convention is x-y-z
ENV ENERGYPLUS_INSTALL_VERSION 8-5-0

# Downloading from Github
# e.g. https://github.com/NREL/EnergyPlus/releases/download
   /v8.3.0/EnergyPlus-8.3.0-6d97d074ea-Linux-x86_64.sh
ENV ENERGYPLUS_DOWNLOAD_BASE_URL https://github.com/NREL/
   EnergyPlus/releases/download/$ENERGYPLUS_TAG
```

```
ENV ENERGYPLUS_DOWNLOAD_FILENAME EnergyPlus-
   $ENERGYPLUS_VERSION-$ENERGYPLUS_SHA-Linux-x86_64.sh
ENV ENERGYPLUS_DOWNLOAD_URL $ENERGYPLUS_DOWNLOAD_BASE_URL/
   $ENERGYPLUS_DOWNLOAD_FILENAME


# Collapse the update of packages, download and
   installation into one command
# to make the container smaller & remove a bunch of the
   auxiliary apps/files
# that are not needed in the container
RUN apt-get update && apt-get install -y ca-certificates
   curl \
    && rm -rf /var/lib/apt/lists/* \
    && curl -SLO $ENERGYPLUS_DOWNLOAD_URL \
    && chmod +x $ENERGYPLUS_DOWNLOAD_FILENAME \
    && echo "y\r" | ./$ENERGYPLUS_DOWNLOAD_FILENAME \
    && rm $ENERGYPLUS_DOWNLOAD_FILENAME \
    && cd /usr/local/EnergyPlus-$ENERGYPLUS_INSTALL_VERSION
        \
    && rm -rf DataSets Documentation ExampleFiles
       WeatherData MacroDataSets PostProcess/
       convertESOMTRpgm \
    PostProcess/EP-Compare PreProcess/FMUParser PreProcess/
       ParametricPreProcessor PreProcess/IDFVersionUpdater

# Remove the broken symlinks
RUN cd /usr/local/bin \
    && find -L . -type l -delete

# Add in the test files
ADD test /usr/local/EnergyPlus-$ENERGYPLUS_INSTALL_VERSION/
   test_run
RUN cp /usr/local/EnergyPlus-$ENERGYPLUS_INSTALL_VERSION/
   Energy+.idd \
        /usr/local/EnergyPlus-$ENERGYPLUS_INSTALL_VERSION/
           test_run/

VOLUME /var/simdata
WORKDIR /var/simdata

CMD [ "/bin/bash" ]
```

Anexo C – Dockerfile – Gromacs

```
# https://bitbucket.org/fourplusone_uni/docker-gromacs/src/
```

```
FROM debian:jessie
RUN apt-get update && apt-get install -y wget

RUN wget ftp://ftp.gromacs.org/pub/gromacs/gromacs-5.0.4.
   tar.gz

RUN echo "c177ae5fd6d71e2bec7369bc66cd082e gromacs-5.0.4.
   tar.gz" > MD5SUM
RUN md5sum -c MD5SUM
RUN tar -xf gromacs-5.0.4.tar.gz

RUN apt-get update && apt-get install -y build-essential
   cmake file libxml2-dev libboost-dev

WORKDIR gromacs-5.0.4
RUN mkdir build
WORKDIR build
RUN cmake .. -DGMX_BUILD_OWN_FFTW=ON -
   DREGRESSIONTEST_DOWNLOAD=ON
RUN make -j `nproc`

# Don't run make check, since mdrun won't work on dockers
   build servers
# RUN make check
RUN make install

WORKDIR /root
RUN echo source /usr/local/gromacs/bin/GMXRC >> .profile

CMD /bin/bash -l
```

Anexo D – Dockerfile – OpenFOAM

```
# https://github.com/qnib/docker-openfoam/blob/12.04_of230/
   Dockerfile

FROM qnib/u_compute:u12.04
MAINTAINER "Christian Kniep <christian@qnib.org>"

RUN echo "deb http://www.openfoam.org/download/ubuntu $(
   lsb_release -cs) main" > /etc/apt/sources.list.d/
   openfoam.list
RUN echo "2014-10-02.1";apt-get update
### openfoam
RUN apt-get install -y libgl1-mesa-glx libgmp10 libqt4-
   opengl libqtcore4 libqtgui4
```

```
RUN apt-get install -y binutils cpp-4.4 g++-4.4 gcc-4.4 gcc
   -4.4-base libstdc++6-4.4-dev
RUN apt-get install -y libibverbs-dev libopenmpi-dev
   openmpi-common  mpi-default-dev
RUN apt-get install -y libc-bin libc-dev-bin libc6 libc6-
   dev libdrm-dev libdrm-nouveau2 libgl1-mesa-dev libglu1-
   mesa libglu1-mesa-dev \
                        libgmp-dev libgmpxx4ldbl libicu48
                           libmpfr-dev libmpfr4 libnuma1
                           libpthread-stubs0 libpthread-
                           stubs0-dev \
                        libqt4-designer libqt4-dev libqt4-
                           help libqt4-opengl-dev libqt4-
                           qt3support libqt4-scripttools
                           libqt4-svg libqt4-test \
                        libqtwebkit-dev libqtwebkit4
                           libtorque2 \
                        libx11-dev libx11-doc libxau-dev
                           libxcb1-dev libxdmcp-dev libxext-
                           dev linux-libc-dev manpages
                           manpages-dev \
                        mesa-common-dev qt4-linguist-tools
                           qt4-qmake \
                        x11proto-core-dev x11proto-input-dev
                            x11proto-kb-dev x11proto-xext-
                           dev xorg-sgml-doctools xtrans-dev
                            zlib1g-dev
RUN apt-get install -y libscotch-5.1 libscotch-dev
RUN apt-get install -y binutils libboost-date-time1.46-dev
   libboost-date-time1.46.1 libboost-dev
RUN apt-get install -y libboost-program-options-dev
   libboost-program-options1.46.1 libboost-serialization1
   .46-dev libboost-serialization1.46.1
RUN apt-get install -y libboost-thread-dev libboost-thread1
   .46-dev libboost1.46-dev

RUN apt-get install -y build-essential
RUN apt-get install -y libptscotch-dev

ADD dpkg /opt/dpkg/
RUN dpkg -i /opt/dpkg/*
RUN apt-get install -y --force-yes openfoam230

RUN sed -i -e 's/   allowSystemOperations.*/
   allowSystemOperations   1;/' $(find /opt/openfoam*/etc -
   name controlDict|head -n1)
```

```
# ENV
RUN echo "source $(find /opt/openfoam*/etc -name bashrc|
   head -n1)" >> /root/.bashrc

CMD supervisord -c /etc/supervisord.conf
```